Data Structures for Java

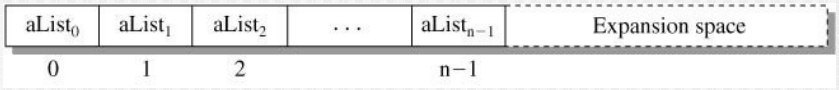
William H. Ford
William R. Topp

Chapter 9reduced
The ArrayList Class
(and clones)

Bret Ford

ArrayList Class

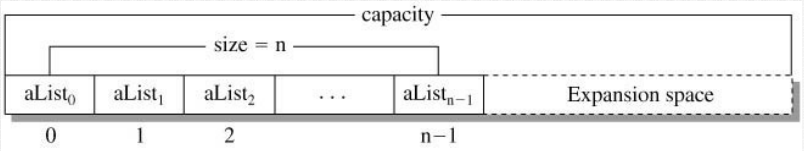
- An ArrayList is a generic structure that stores elements in a contiguous block of memory. The back of the list provides expansion space that can grow as new elements are added.



The methods get(i) and set(i, item) use direct access (O(1)) to access and update an element in the list.

ArrayList Sizing

- The capacity is the number of elements in the block of memory that hold the list. The size is the number of elements currently in the ArrayList.



- Method ensureCapacity(minCapacity) insures that the storage block has at least minCapacity available elements.
- Method trimToSize() trims the capacity of the list to its current size.

ArrayList API

class ARRAYLIST<T> implements LIST<T>	
	Constructors
	ArrayList() Creates an empty ArrayList.
	Methods
void	**ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

class ARRAYLIST<T> implements LIST<T>	
String	**toString**() Returns a string listing the elements in the ArrayList as comma separated values enclosed in brackets.
void	**trimToSize**() Trims the capacity of this ArrayList to be the ArrayList's current size.

Joining ArrayLists

The method join(listA, listB) concatenates listB onto the back of listA

```
public static <T> void join (ArrayList<T> listA,
ArrayList<T> listB)
{
   // capture the size of ArrayLists listA and listB
   int i, sizeA = listA.size(), sizeB = listB.size();

   // insure sufficient capacity for listA
   listA.ensureCapacity(sizeA + sizeB);

   // use index i to access the elements of listB and
   // add() to insert elements from listB at rear of listA
   for (i = 0; i < sizeB; i++)
      listA.add(listB.get(i));
}
```

Program 9.1

The program inputs a list of graduates (BA and BS) and uses join() to create an ArrayList with the names of BS graduates followed by the names of BA graduates.

```
import java.util.Scanner;
import java.io.*;
import ds.util.ArrayList;
public class Program9_1
{
   public static void main(String[] args)
   throws IOException
   {
      Scanner fileIn = new Scanner(
         new FileReader("gradlist.dat"));
```

Program 9.1 (continued)

```
// input strings from the file
String inputStr, gradName, gradDegree;
// string of 20 blank characters for ArrayList names
String buffer = "                    ";
// ArrayLists holds diplomaList and baList
ArrayList<String> diplomaList = new ArrayList<String>(),
                  baList = new ArrayList<String>();
// delimiters are tab, newline, and carriage return
fileIn.useDelimiter("[\t\n\r]+");

// read registrar's list to eof and add to array lists
while(fileIn.hasNext())
{
   // input tab separated name and degree
   gradName = fileIn.next();
   // input the degree
   gradDegree = fileIn.next();
```

Program 9.1 (continued)

```
      // add name and degree as
      // string in specified list
      if (gradDegree.equals("BS"))
         diplomaList.add(gradName + "  " +
                         gradDegree);
      else
         baList.add(gradName + "  " +
                    gradDegree);
   }

   // join the BA list at end of diploma list
   join(diplomaList, baList);
```

Program 9.1 (concluded)

```
      // output a header and list of names with degrees
      System.out.println("Diploma List");
      for (int i = 0; i < diplomaList.size(); i++)
         System.out.println("\t" +
               (String)diplomaList.get(i));
   }

   < method join() provided in the program discussion >
}
```

Program 9.1 (File “gradlist.dat”)

```
Bailey, Julie BS
Frazer, Thomas BA
Harkness, Bailey BA
Johnson, Shannon BS
Kilmer, William BA
Miller, Sara BS
Nelson, Harold BS
O'Dell, Jack BA
Wilson, Rebecca BS
```

Program 9.1 (Run)

```
Diploma List
   Bailey, Julie BS
   Johnson, Shannon BS
   Miller, Sara BS
   Nelson, Harold BS
   Wilson, Rebecca BS
   Frazer, Thomas BA
   Harkness, Bailey BA
   Kilmer, William BA
   O'Dell, Jack BA
```

closestPair()

Returns the indices of the two closest points in set of points whose coordinates are in ArrayLists x and y

```
public static int[]
closestPair(ArrayList<Double> x, ArrayList<Double> y)
{
   // capture the number of points in n
   int n = x.size();
   // index1 and index2 will contain the
   // indices of the closest points
   int i,j, index1 = -1, index2 = -1;
   double xi, yi, xj, yj;
   // initialize dmin to the largest possible double value
   double dmin = Double.MAX_VALUE, dsqr;
   // we return this array after determining its values
   int[] closest = new int[2];
```

closestPair() (continued)

```
// make n-1 passes through the points
for (i=0; i < n-1; i++)
   // compute each distance d(Pi, Pj),
   // i+1 <= j < n and record the current
   // minimum distance
   for (j=i+1;j < n;j++)
   {

      // extract the double values from x and y
      xi = x.get(i);
      yi = y.get(i);
      xj = x.get(j);
      yj = y.get(j);
      // compute (xi - xj)^2 + (yi - yj)^2
      dsqr = sqr(xi - xj) + sqr(yi - yj);
```

closestPair() (concluded)

```
         // check for a new minimum distance
         if (dsqr < dmin)
         {

            // new minimum; record it and change
            // indices index1 and index2
            dmin = dsqr;
            index1 = i;
            index2 = j;
         }
      }

   // initialize the elements of closest[] and return it
   closest[0] = index1;
   closest[1] = index2;
   return closest;
}
```

Program 9.2

```
import ds.util.ArrayList;
public class Program9_2
{
   public static void main(String[] args)
   {
      // insert coordinates from a file into x and y
      ArrayList<Double> x = new ArrayList<Double>(),
                        y = new ArrayList<Double>();
      double xCoor, yCoor;
      double xclose1, yclose1, xclose2, yclose2;
      // arrays for x-points and y-points and closest points
      double[] xPt = {-1, -1.45, -1, -1, 0.75, 1, 0.8, 1.65,
                      -1.2, 3, 2, 2.7, 1.35, 0.5};
      double[] yPt = {1, 1.3, 0, 3, -1.75, -1, 0.7, 1, -1.3,
                       0, 2, 1.3, 1.1, 2.45};
```

Program 9.2 (continued)

```
      int[] closestPoints;

      // add coordinate values at the back of x and y
      for (int i = 0; i < xPt.length; i++)
      {
         x.add(xPt[i]);
         y.add(yPt[i]);
      }

      // execute the closest-pair algorithm
      closestPoints = closestPair(x, y);

      // find the coordinates of the closest points
      xclose1 = x.get(closestPoints[0]);
      yclose1 = y.get(closestPoints[0]);
      xclose2 = x.get(closestPoints[1]);
      yclose2 = y.get(closestPoints[1]);
```

Program 9.2 (concluded)

```
      // output pair of points and their minimum distance
      System.out.println("The closest points are ("
         + xclose1 + "," + yclose1 + ") and
         (" + xclose2 + "," + yclose2 + ")");
      System.out.println("Distance = " +
         Math.sqrt(sqr(xclose1-xclose2) +
                   sqr(yclose1-yclose2)));
   }
   < implementation of closestPair() and
     sqr() given in the program discussion >
}


      Run:

      The closest points are (1.65,1.0) and (1.35,1.1)
      Distance = 0.31622776601683783
```

Implementing ArrayList

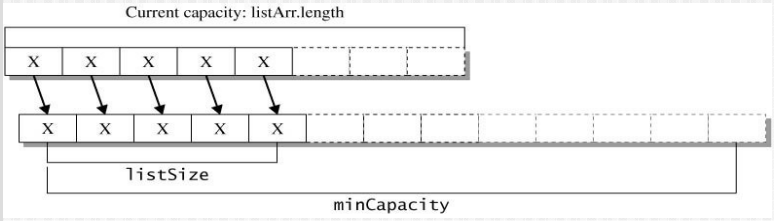
An ArrayList object uses an array of specified type to store the elements. Instance variables define the array and the size. The constructor allocates an array which has 10 elements. This is the initial capacity for the array.

Implementing ArrayList (variables and constructor)

```
public class ArrayList<T> implements List<T>
{
   private T[] listArr;    // stores the elements
   private int listSize;   // number of elements in the list

   // constructs an empty list with initial capacity 10
   public ArrayList()
   {
      listArr = (T[])new Object[10];
      listSize = 0;
   }
   . . .
}
```

ensureCapacity()




```
public void ensureCapacity (int minCapacity)
{
   // get the current capacity
   int currentCapacity = listArr.length;
   // only take action if the requested capacity
   // is larger than the existing capacity
   if (minCapacity > currentCapacity)
   {
      // capture a reference to the old array
      T[] oldListArr = listArr;
```

ensureCapacity()

```
      // create the new array with the new capacity
      listArr = (T[]) new Object[minCapacity];

      // copy the old data to the new array
      for (int i=0; i < listSize; i++)
         listArr[i] = oldListArr[i];

      // nullify reference to the old array; garbage
      // collection will recover the space
      oldListArr = null;
   }
}
```

Private rangeCheck() Method

The index-access methods in a List class call the method rangeCheck() to check whether the index is in range 0 to size()-1.

```
// verify that index is in the range
// 0 <= index <= upperBound; if not throw
// the IndexOutOfBoundsException exception
private void rangeCheck(int index, String msg,
int upperBound)
{
   if (index < 0 || index >= upperBound+1)
      throw new IndexOutOfBoundsException(
         "\n" + msg + ": index " + index +
         " out of bounds. Should be in the range 0 to " +
         upperBound);
}
```

add(index, item) Method

```
public void add(int index, T item)
{
   // index == listSize is valid; append to the list
   rangeCheck(index, "ArrayList add()", listSize);

   // see if we need to reallocate more memory
   if (listSize == listArr.length)
      ensureCapacity(2*listArr.length);

   // shift elements at index through listSize-1 to the right
   for (int j=listSize-1;j >= index;j--)
      listArr[j+1] = listArr[j];

   // insert item at location index and increment listSize
   listArr[index] = item;
   listSize++;
}
```

add(item) Method

```
// appends item to the end of this
// list and returns true
public boolean add(Object item)
{
   // call method add() at an index to
   // insert item at the end of the list
   add(listSize, item);

   return true;
}
```

Method has running time O(1).

remove(index) Method

```
public T remove(int index)
{
   // verify that index is in the proper range
   rangeCheck(index, "ArrayList remove()",
              listSize-1);

   // save the return value
   T returnElement = listArr[index];

   // shift elements at indices index+1 to listSize-1
   // left one position
   for (int j=index;j < listSize-1;j++)
      listArr[j] = listArr[j+1];
```

remove(index) Method (2)

```
      // make former last entry a null
      // reference and decrement list size
      listArr[listSize-1] = null;
      listSize--;

      // return the value that was removed
      return returnElement;
   }
```

Method has average running time O(n).

remove(item)

```
// if item is present in the list, removes the first
// instance of it from this list; returns true if
// an element was removed and false otherwise
public boolean remove(Object item)
{
   int i = 0, j;
   boolean retValue = true;
   // use indexOf() to search for item
   if ((i = indexOf(item)) != -1)
      remove(i);
   else
      retValue = false;

   return retValue;
}
```

Method has average running time O(n).

get()

```
public T get(int index)
{
   // verify that index is in the proper range
   rangeCheck(index, "ArrayList get()", listSize-1);

   return listArr[index];
}
```

Method has direct access with running time O(1).

set()

```
// replaces the value at the specified position with item
// and returns the previous value; if index is out of range
//index<0 o index>=size,throws IndexOutOfBoundsException
public T set(int index, T item)
{
   // verify that index is in the proper range
   rangeCheck(index, "ArrayList set()", listSize-1);
   // save the element at listArr[index]
   T previousValue = listArr[index];
   // assign the new element at index index
   listArr[index] = item;
   // return the previous element
   return previousValue;
}
```

Method has direct access with running time O(1).

Cloneable Objects

- A clone object is a copy of the existing object. The cloning process creates a new object with a field-for-field copy of values from the instance variables in the existing object.
- Instances of a class can be cloned provided the class defines the public method clone() and implements the Cloneable interface.

E.g.
```
public class Demo implements Cloneable
{
    public Object clone()
    . . .
}
```

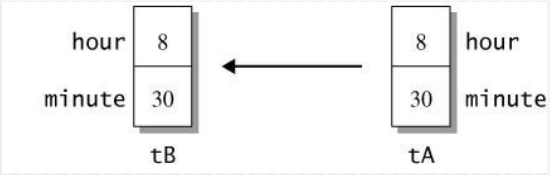
Implementing clone()

- In a class, implement the clone() method by defining an Object instance variable and assigning it the return value from the protected Object method clone(). The instance variable references a copy of the object and is assigned as the return value for clone().

E.g.
```
public Object clone()
{
    Object copy = null;
    copy = (Demo)super.clone();
    return copy;
}
```

Cloneable Time24 Class

Use:
```
Time24 tA = new Time24(8, 30), tB;
tB = (Time24)tA.clone();
```




```
public class Time24 implements Comparable<Time24>,
Cloneable
{
   ...
   public Object clone()
   {
      // the clone that is returned
      Object copy = null;
```

Cloneable Time24 Class (2)

```
        try
        {
           // call the Object method clone();
           // copy is a reference to a Time24 object
           copy = (Time24)super.clone();
        }
        catch (CloneNotSupportedException cnse)
        {
           // exception indicates a fatal error in
           // the virtual machine
           throw new InternalError();
        }

        return copy;
     }
  }
```

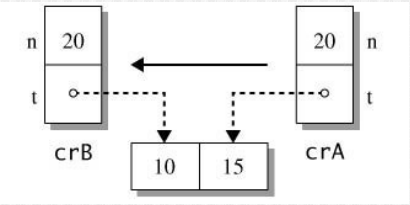
Cloning Reference Variables

- Method clone() in the Time24 class creates a new object with primitive variables for hour and minute with the same values as the original object.
- Updates to either object do not affect the other.

```
Time24 tA = new Time24(8, 30), tB;
tB = (Time24)tA.clone();
tA.addTime(15);    // tA = 8:45 tB = 8:30
```

Cloning Reference Variables (2)

- When a class has a reference variable, cloning an object creates a new copy of the variable. However, the new variable references the same object as the original variable.
- E.g. CloneRef class variables are a primitive int n and a Time24 reference t. Assume crB is a clone of crA.

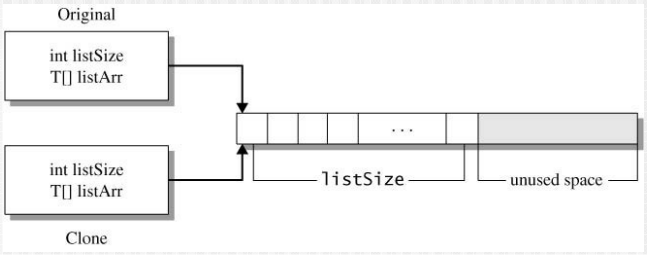


CloneRef Class

```
class CloneRef implements Cloneable
{
   private int n;
   private Time24 t;
   ...
   public Object clone()
   {
      Object copy = null;
      try
      { copy = super.clone(); }
      catch (CloneNotSupportedException cnse)
      { throw new InternalError(); }
      // return the cloned object
      return copy;
   }
   ...
}
```

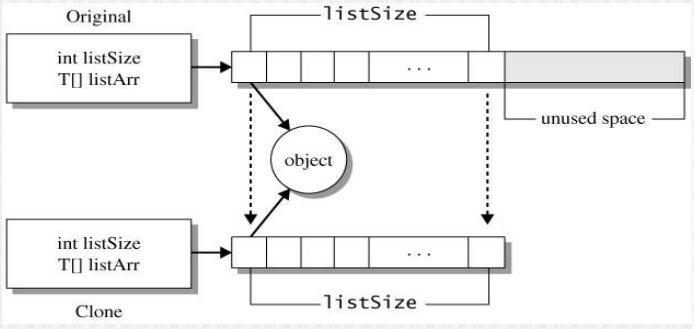
Cloning an ArrayList (1)

- The ArrayList class has a primitive int listSize and an array reference listArr. Implement clone() by using Object class clone() to create new variables. Then allocate an array for the clone with length listSize and copy elements from original array to newly allocated array.



Cloning an ArrayList (2)

```
// replace listArr in copy by a new reference to an array
copy.listArr = (T[])new Object[listSize];
// copy the elements from listArr to copy.listArr
for (int I = 0; I < listSize; i++)
     copy.listArr[i] = listArr[i];
```



Cloning an Array

```
// create a Time24 array and a clone
Time24[] timeArr = {new Time24(7,15), new Time24(14,00),
                    new Time24(3,45), new Time24(12,30)};
Time24[] cloneTimeArr = timeArr.clone();

System.out.println("Original: timeArr[0] = " +
                   timeArr[0] + "  cloneTimeArr[0] = "
                   + cloneTimeArr[0]);

// update timeArr[0] by advancing time 30 minutes
timeArr[0].addTime(30);

// display value at index 0 in timeArr and its clone
System.out.println("Updated:  timeArr[0] = " +
                   timeArr[0] + "  cloneTimeArr[0] = "
                   + cloneTimeArr[0]);
```

Cloning an Array(2)

```
// sort cloneTimeArr
Arrays.sort(cloneTimeArr);

// display the elements for the two arrays
System.out.println("Unsorted timeArr:    " +
                   Arrays.toString(timeArr));
System.out.println("Sorted cloneTimeArr: " +
                   Arrays.toString(cloneTimeArr));
```

```
Output:

Original: timeArr[0] = 7:15  cloneTimeArr[0] = 7:15
Updated:  timeArr[0] = 7:45  cloneTimeArr[0] = 7:45
Unsorted timeArr:    [7:45, 14:00, 3:45, 12:30]
Sorted cloneTimeArr: [3:45, 7:45, 12:30, 14:00]
```