Arrays vs. ArrayList Class

Array

Collection mechanism common to most modern programming languages.

The programmer has more control over the amount of memory used, but requires some understanding of memory allocation.

Because memory access is not protected by the services of a class, memory access errors are commonly encountered by beginning programmers.

Can have arrays of objects and arrays of primitive types.

Casting is not required.

ArrayList

Implemented as a class, so it is easy to use the "services" provided, such as **add()**, **remove()** and **contains()**.

Hides many implementation details which can be useful for the beginner, but the understanding of these details are important to students of Computer Science.

Unique to Java; may not be available in other languages.

Handles only objects; requires casting.

Two Short Programs

Using an Array

```
public static void main(String[] a)
{  String[ ] movies;

   movies = new String[6];

   movies[0] = "Wizard of Oz";
   movies[1] = "Terminator 2";
   movies[2] = "Shrek";
   movies[3] = "Moulin Rouge";
   movies[4] = "American Beauty";
   movies[5] = "The Color Purple";

   for(int i=0; i<movies.length; i++)
   {  String m = movies[i];

      System.out.println(m);
   }
}
```

Using an ArrayList

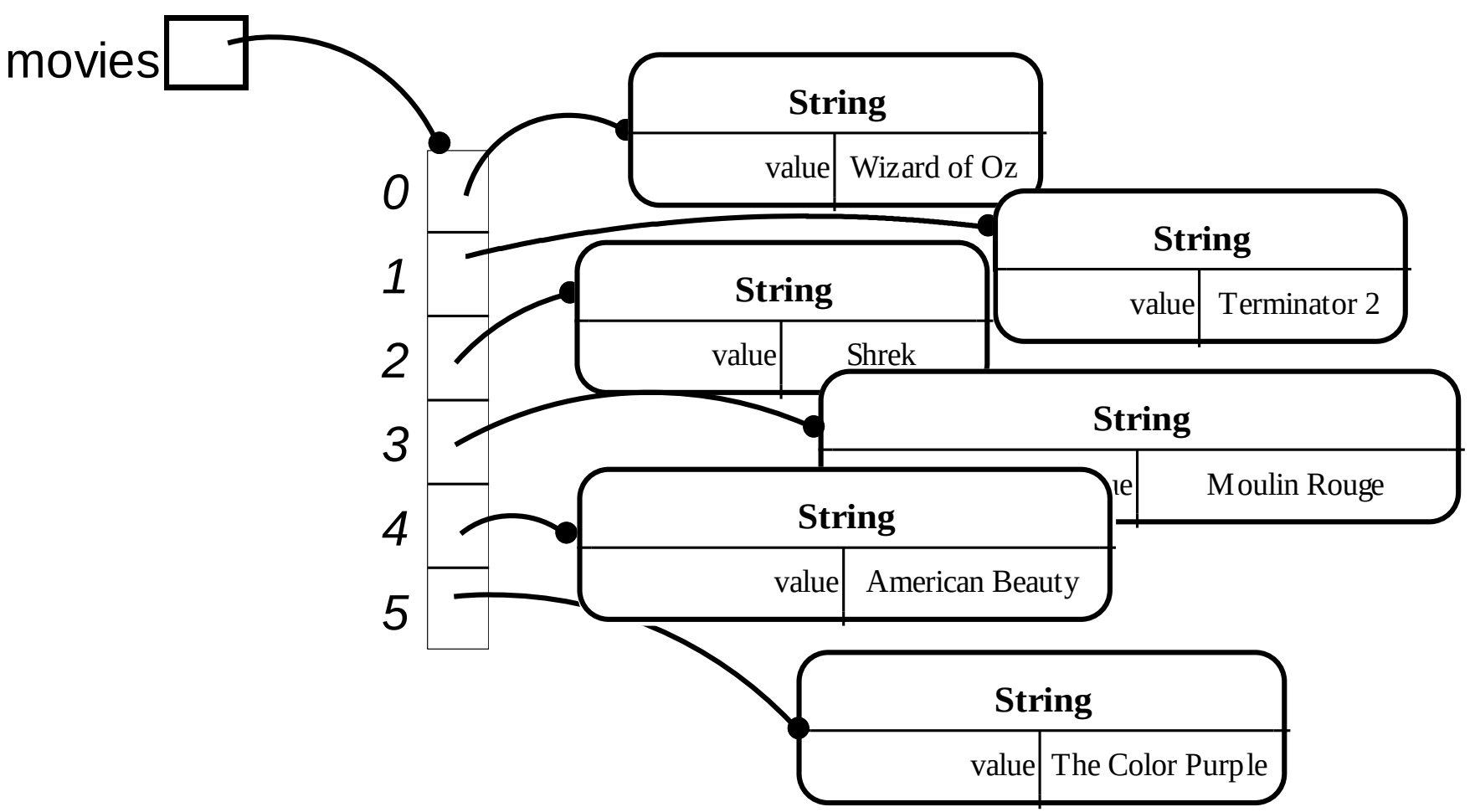
```
public static void main(String[] a)
{  ArrayList movies;

   movies = new ArrayList();

   movies.add("Wizard of Oz");
   movies.add("Terminator 2");
   movies.add("Shrek");
   movies.add("Moulin Rouge");
   movies.add("American Beauty");
   movies.add("The Color Purple");

   for(int i=0; i<movies.size(); i++)
   {  String m =
            (String)movies.get(i);
      System.out.println(m);
   }
}
```

Visusalizing an Array



Common Array Error #1

```
public class MovieList extends Object
{  private String[ ] bestMovies;

   /** Construct a list of best movies. */
   public MovieList()
   {  super();
      this.bestMovies = new String[5];
      this.bestMovies[1] = "Terminator 2";
      this.bestMovies[2] = "Shrek";
      this.bestMovies[3] = "Dude, Where's My Car?";
      this.bestMovies[4] = "American Beauty";
      this.bestMovies[5] = "Wizard of Oz";
   }

   /** Get the movie at position i. */
   public String getMovie(int pos)
   {  if (pos < 1 || pos > this.bestMovies.length)
      {  throw new Error("Invalid Position; pos = " + pos);
      }
      return this.bestMovies[pos];
   }
}
```

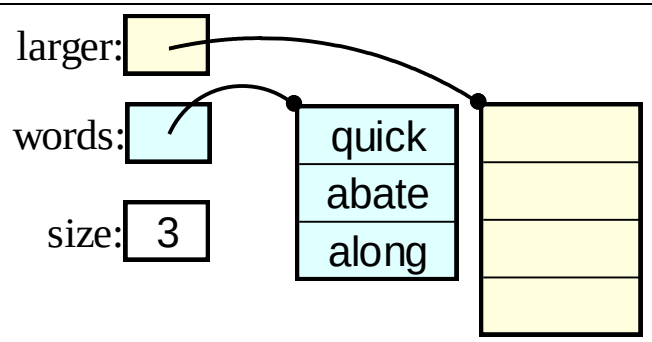
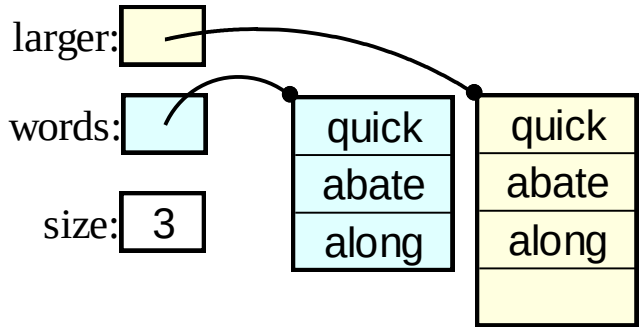
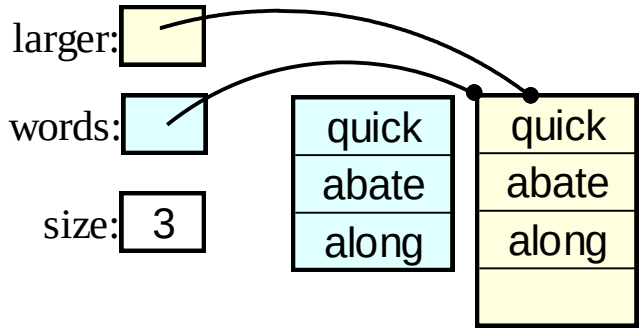
Common Array Error #2

```
public class MessageBuffer extends Object
{  private String[ ] lines;

   /** Construct a list of Strings. */
   public MessageBuffer ()
   {  super();
      this.lines = new String[100];
   }

   /** Change the message to all caps.*/
   public void shout()
   {  for(int pos = 0; pos < this.lines.length; pos++)
      {  this.lines[pos] = this.lines[pos].toUpperCase();
      }
   }
}
```

Enlarging an Array

Step 1: Allocate a new, larger array	
Step 2: Copy from old array to new	
Step 3: Reassign references	

Adding a Word to the Array

```
public class FlexibleList extends Object
{  private String[ ] words = new String[0];

   public FlexibleList(String fileName)
   {  TextInput in = new TextInput(fileName);
      while (!in.eofIsAvailable())
      {  this.addWord(in.readToken());
      }
      in.close();
   }

   public void addWord(String w)
   {  String[ ] larger = new String[this.words.length + 1];
      for(int i=0; i<this.words.length; i++)
      {  larger[i] = this.words[i];
      }
      larger[larger.length-1] = w;

      this.words = larger;
   }
}
```

How Much "Work" for the Computer?

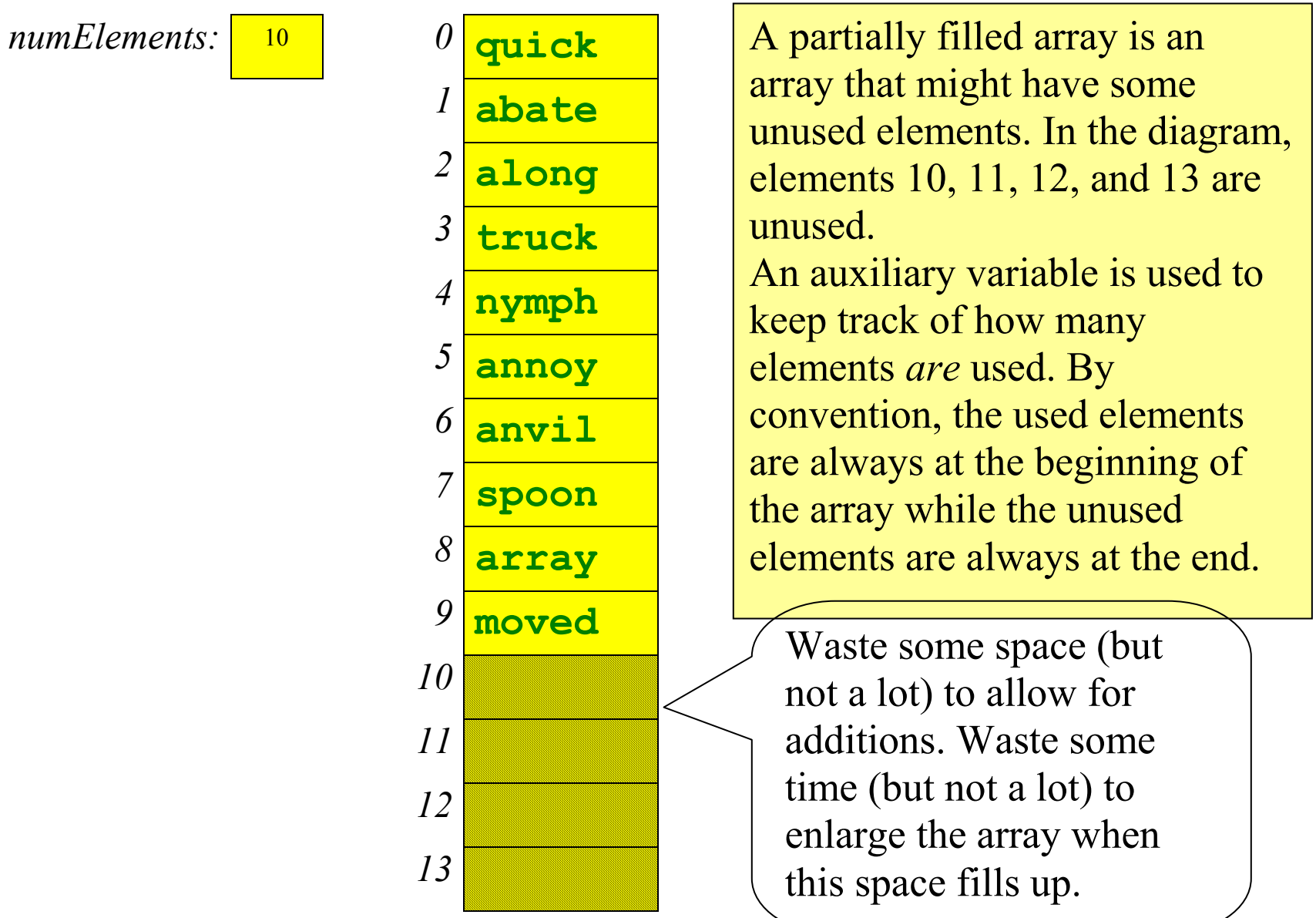
	Grow Array by 1			
Insert Word	New Size	Copies	Insert	Total Work
1	1	0	1	1
2	2	1	1	3
3	3	2	1	6
4	4	3	1	10
5	5	4	1	15
6	6	5	1	21
7	7	6	1	28
8	8	7	1	36
9	9	8	1	45
10	10	9	1	55
11	11	10	1	66
$n = 2^i$	n	$n - 1$	1	

How much "work" is done to insert n words?

n	**"work"**
1,024	524,800
2,048	2,098,176
4,096	8,390,656
8,192	33,558,528
16,384	134,225,920
32,768	536,887,296
65,536	2,148,499,456
131,072	8,590,000,128

Too Much Work!

Partially-Filled Array



A partially filled array is an array that might have some unused elements. In the diagram, elements 10, 11, 12, and 13 are unused.

An auxiliary variable is used to keep track of how many elements *are* used. By convention, the used elements are always at the beginning of the array while the unused elements are always at the end.

Waste some space (but not a lot) to allow for additions. Waste some time (but not a lot) to enlarge the array when this space fills up.

Growing the Array

```
public class BetterFlexibleList extends Object
{  private static final int INITIAL_CAPACITY = 3000;
   private String[ ] words = new String[INITIAL_CAPACITY];
   private int numWords = 0;

   public BetterFlexibleWordList(String fileName)
   {  …   }

   // Enlarge the list of words to make room for some more.
   private void growWordList()
   {  String[ ] larger = new String[ this.words.length()*2 + 1 ];

      for(int i=0; i< this.words.length(); i++)
      {  larger[i] = this.words[i]
      }

      this.words = larger;
   }
}
```

Adding an element

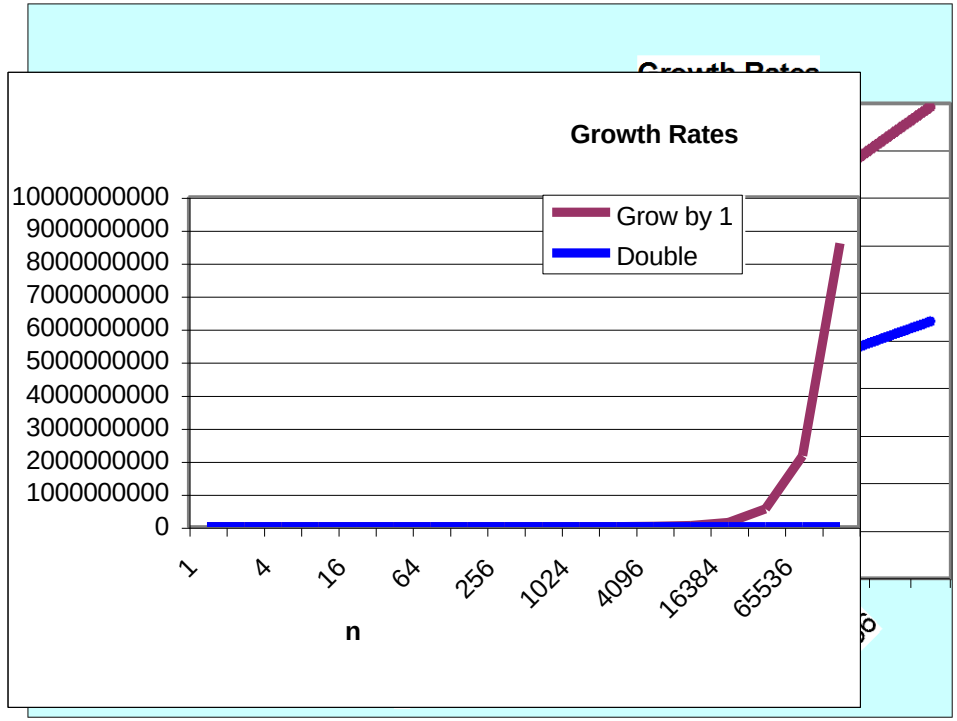
```
if (this. numWords >= this.words.length)
{  this.growWordList();
}
this.words[this.numWords] = aWord;
this.numWords++;
```

Growth Analysis

	Grow Array by 1				Double Array			
Insert Word	New Size	Copies	Insert	Total Work	New Size	Copies	Insert	Total Work
1	1	0	1	1	1	0	1	1
2	2	1	1	3	2	1	1	3
3	3	2	1	6	4	2	1	6
4	4	3	1	10			1	7
5	5	4	1	15	8	4	1	12
6	6	5	1	21			1	13
7	7	6	1	28			1	14
8	8	7	1	36			1	15
9	9	8	1	45	16	8	1	24
10	10	9	1	55			1	25
11	11	10	1	66			1	26
$n = 2^i$	n	$n - 1$	1	n(n +1)/2	n	n/2	1	2n -1

Grwoth Rates

n	Grow by 1	Double
1,024	523,776	2,047
2,048	2,096,128	4,095
4,096	8,386,560	8,191
8,192	33,550,336	16,383
16,384	134,209,536	32,767



Summary

- Arrays can be a collection of any primitive or object type.
- Size of the array must be specified before it can be used (unlike **ArrayList**).
- Individual items in the collection are referenced with **[ ]**; the entire collection is referred to by the variable name.
- Individual items referenced with **[ ]** can be used in the same way that the base type is used—as a parameter (either explicit or implicit), as part of an expression, etc.
- One common error is to forget to fill the array with objects before accessing the objects.
- You can reassign the array reference to an array of a different size.
- When resizing arrays, doubling the size is much more efficient (in terms of processing time) than increasing the size by one.

These slides are based upon notes from CS132 (Winter 2005) at the University of Waterloo.