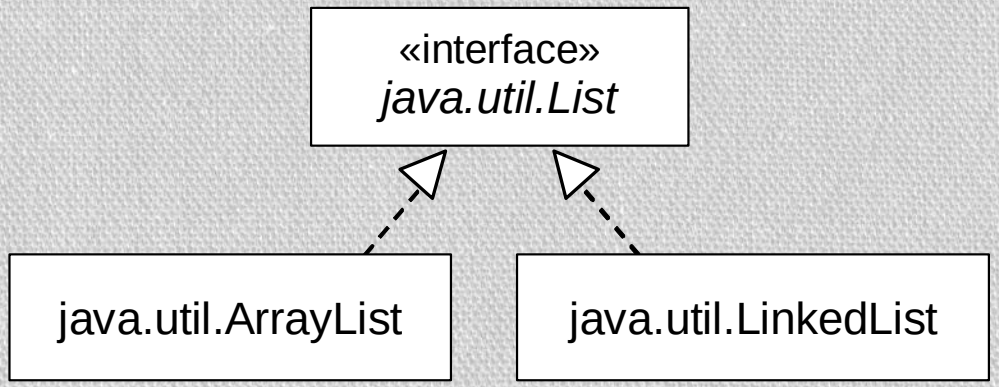
java.util.ArrayList

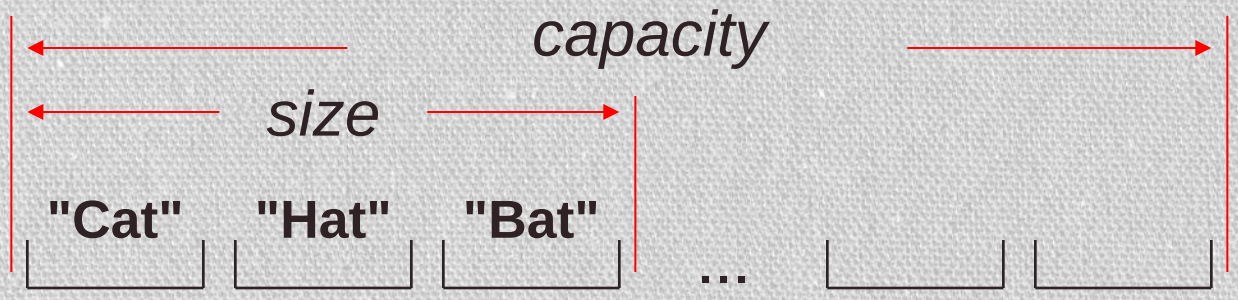
java.util.ArrayList<*E*>

- Implements a list using an array
- Implements java.util.List<*E*> interface



java.util.ArrayList<*E*> cont'd

- Implements a list using an array.
- Can only hold objects (of a specified type), not elements of primitive data types.
- Keeps track of the list *capacity* (the length of the allocated array) and list *size* (the number of elements currently in the list)



ArrayList

- The elements' data type is shown in angle brackets and becomes part of the List and ArrayList type. For example:

```
ArrayList<String> words = new ArrayList<String>();

List<Integer> nums = new ArrayList<Integer>();
```

ArrayList<E> Constructors

Java docs use the letter ***E*** as the type parameter for elements in generic collections

ArrayList<E> ( )

ArrayList<E> (int capacity)

Creates an empty **ArrayList<*E*>** of default capacity (ten)

Creates an empty **ArrayList<*E*>** of the specified capacity

ArrayList<*E*> Methods

int **size**()

boolean **isEmpty** ()

boolean **add** (E obj)

void **add** (int i, E obj)

E **set**(int i, E obj)

E **get**(int i)

E **remove**(int i)

boolean **contains**(E obj)

int **indexOf**(E obj)

returns **true**

inserts **obj** as the **i**-th value; **i** must be from 0 to **size()**

i must be from 0 to **size() –1**

both use **equals** to compare objects

ArrayList Example

```
ArrayList<String> names =
                    new ArrayList<String>( );
names.add("Ben");
names.add("Cat");
names.add(0, "Amy");
System.out.println(names);
```

Output

[Amy, Ben, Cat]

ArrayList's **toString** method returns a string of all the elements, separated by commas, within **[ ]**.

ArrayList<*E*> Details

- Automatically increases (doubles) the capacity when the list runs out of space (allocates a bigger array and copies all the values into it).
- get(i) and set(i, obj) are efficient because an array provides random access to its elements.
- Throws IndexOutOfBoundsException when

  $i < 0$ or $i \geq$ size()

  (or $i >$ size() in add (i, obj))

ArrayList<*E*> Autoboxing

- If you need to put integers or doubles into a list, use a standard Java array or convert them into Integer or Double objects
- Since Java 5, conversion from int to Integer and from double to Double is, in most cases, automatic (a feature known as *autoboxing* or *autowrapping*); the reverse conversion (called *autounboxing*) is also automatic.

ArrayList<*E*> Autoboxing Example

```
ArrayList<Integer> counts =
                    new ArrayList<Integer>( );

counts.add(17);
...
int count = counts.get(0);
```

Autoboxing: compiled as **counts.add(new Integer(17));**

Autounboxing: **count** gets the value 17

ArrayList Deficiencies

```
// Remove all occurrences
// of "like" from words:

int i = 0;

while (i < words.size())
{
   if ("like".equals(words.get(i))
      words.remove(i);
   else
      i++;
}
```

Caution: when you remove elements, a simple **for** loop doesn't work:

```
for (int i  = 0; i < words.size(); i++)
{
    if ("like".equals(words.get(i))
       words.remove(i);
 }
```

Shifts all the elements after the **i**-th to the left and decrements the size

“For Each” Loop

```
ArrayList<String> words = new ArrayList<String> ( );
...
for (String word : words)
{
    ...  // process word
}
...
```

The same as:

```
for (int i = 0;  i < words.size ();  i++)
{
    String word = words.get (i);
    ... // process word
}
```