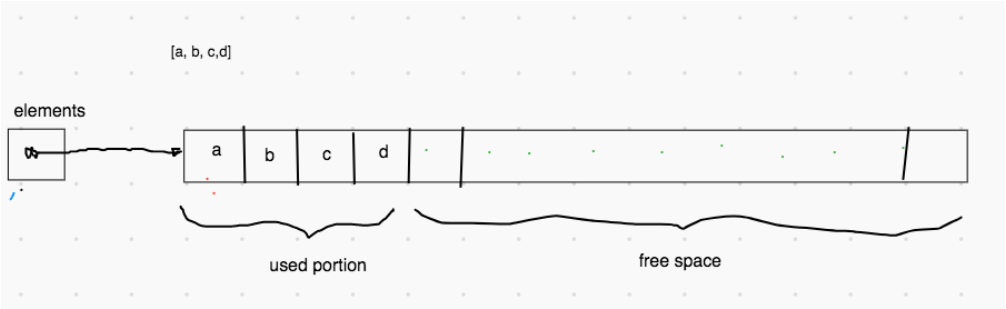
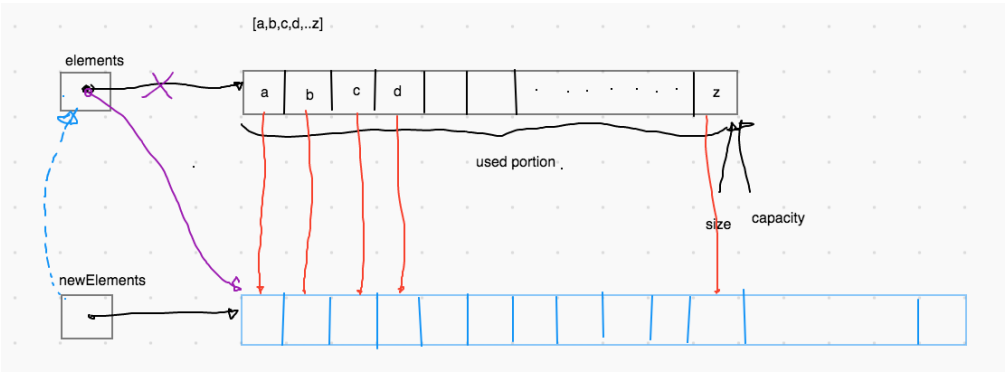
ArrayList

A list implementation using arrays.

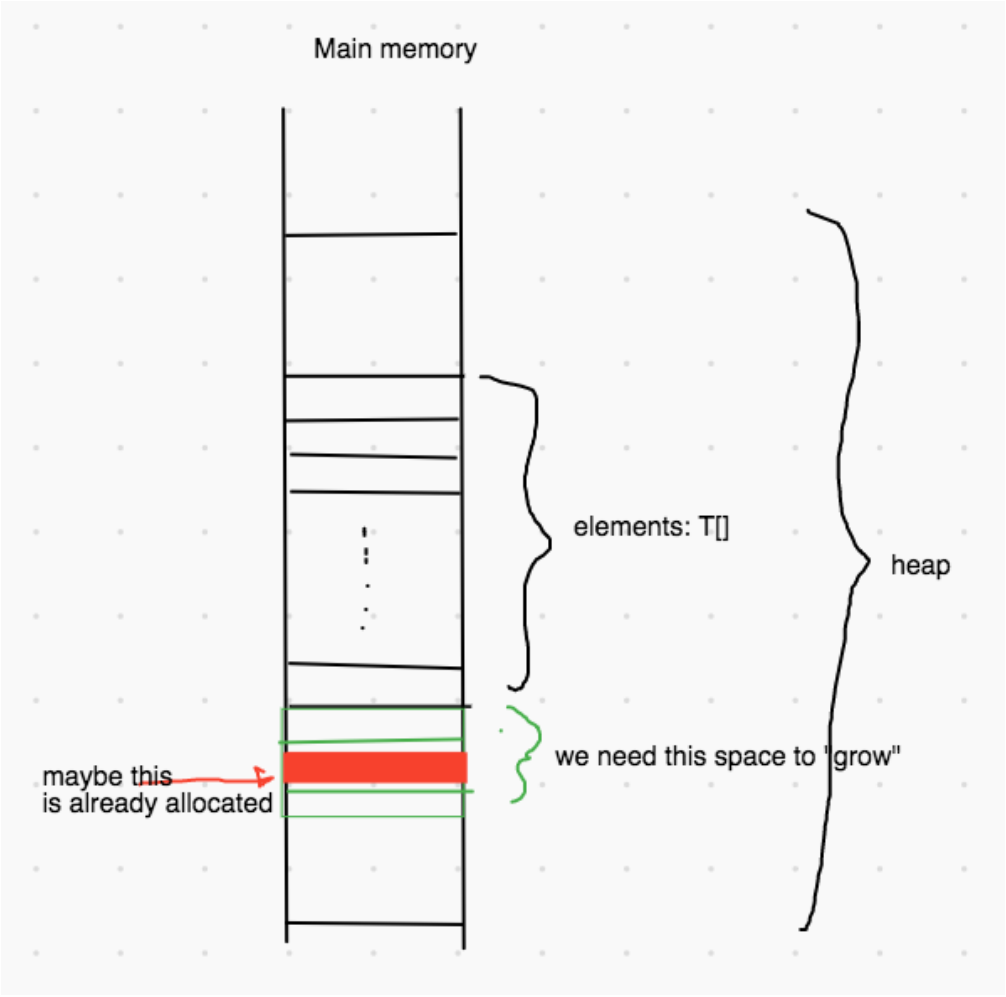


Capacity considerations

To implement the list API in a comparable way to the `LinkedList` we should not run out of cells to put data, i.e.: there is no ”capacity”. Unless we do something, we will eventually run out of space in our array:



To add more cells to the allocated array isn’t possible, since the adjacent memory in the heap might not be available:



We need to ”grow” the array by allocating a bigger array and copying over all the elements. Here is an implementation:

```
private void checkCapacityAndExpand() {
    if(size < elements.length)
        return;
    int newCapacity = ...;  // TODO
    T[] newArray = (T[]) new Object[newCapacity];
    for (int i = 0; i < size; i++)
        newArray[i] = elements[i];
    elements = newArray;
}
```

In class we explored some ideas for the TODO above. Here are some results from populating an `ArrayList` with 50000 elements:

Grow by	Calls to expand	Element moves while expanding	Available spaces
1	49991	1,250,024,955	0
100	500	12,480,000	9
35%	40	246817	11696
50%	22	142273	21139
100% (x2)	13	81910	31919
cap^2	3	10110	99949999

Observations:

- Increasing by a constant number keeps the unused portion of the array low, but results in more expands and more moves.
- Increaing by a percent lowers the number of calls to expand, so less moves, but with a potential for unused space.

Usually we think of 25%-100% increase as reasonable conpromize.

Other considerations:

1. Allow the user to specify the increase amount.
2. Set the initial capacity to sufficient.
3. Add a ”trim” feature to the array list to reduce the capacity in the end.