CS136: Data Structures & Advanced Programming

Fall 2020
Williams College

Today

Deep dive into `Vector` class, including:

- Where to find the source code
- Important implementation details
- Usage (especially when compared to arrays)

Vectors: Examining the Code

We'll focus on Structure5:

- Code associated with the textbook is publicly available
    - bailey.jar - archive used by javac
    - structure-source.tgz - compressed bundle of Java text files
        - After uncompressing, `src/structure5` has the code we want!
- Javadoc for the code is publicly available too!

Vector<E> API (select methods)

- `get(int), set(int, E)`
- `firstElement(), lastElement()`
- `contains(E), indexOf(E)`
- `add(E), addElement(E), add(int,E)`
- `remove(E)`
- `clear()`
- `capacity()`
- `ensureCapacity()`
- `toString()`

Vector Details: Storing Data

Internally, the `Vector` class stores an array: `Object[] elements;`

- The array is not necessarily filled
- We keep track of the number of current elements in the array using an explicit `elementCount` variable
  - How do we ensure that `elementCount` stays in sync with our actual count?
  - What happens if we try to add an element but the array is full?
- Overloaded constructor(s) allow us to specify an initial array size (we'll call this the Vector's capacity)
  - Default capacity used if none is provided

Vector Details: get(int)/set(int, E)

Arrays use bracket notation to access and update elements at a given index. Vectors use methods.

- We can’t use bracket notation for non-array objects. We must call methods. But *internally*:
  - `v.get(int)` uses bracket notation to *access* `elementData[i]`
  - `v.set(int, E)` uses bracket notation to *update* `elementData[i]`

Get/set cost is the same as the cost of accessing/updating an array.

Vector Details: add(E)

Arrays don't have any notion of "appending". add(E) is "Vector append"

- What does it mean to "append" to a Vector?

When we think about performance, we often care most about the "worst case"

- What are the "worst cases" that we need to consider when appending to a Vector?
  - If the Vector's internal array has room, we can just place the element at the first free index, and increment the count
  - If the Vector's internal array is full, we need to GROW! This means creating a larger array, copying everything into it, then adding the new element.
    - How big should we make the new array?

Vector Details: add(int, E)

Arrays don't have any notion of "inserting". add(int, E) inserts at index i

- What does it mean to insert into the middle of a Vector?

Unlike an array that overwrites the element at a given index, a Vector "creates room", then adds the element in that newly emptied space

- How do we create room in the Vector's internal array?
  - Shift all elements *after* the insertion point one space to the right

Vector Details: contains(E)

contains(E) determines if a value appears in the Vector

- What does it mean for a value to "appear in" a Vector?
  - elementData[i].equals(obj) == true   (for some index i)
  - *Note:* indexOf(E) is similar, except it returns the index i, or -1 if not found

- What if there are multiple copies of the target value?

  - No worries! We just return true as soon as we find the first occurrence

- Note that contains uses .equals, and we can only call .equals on Objects.

  - We can't store primitive values in an array!

Vector Details: remove(E)

remove(E) removes the first occurrence of a value from the Vector

- Similar to contains: search using .equals to find a match
- What if there are multiple copies of the target value?
    - Delete the first. We stop as soon as we remove the first occurrence

Vector Details: size()

Vector size is different than Vector capacity.

- Size is how many elements are *currently* in the underlying Object array
- Capacity is the length of the underlying array
- How do they differ?
    - The array may not be full! (Note: size <= capacity)
    - As we add and delete elements, size will fluctuate, but array size cannot change.
    - We may “grow” or “shrink” our array by creating a new array and copying items
        - When/how we do this has huge implications on performance! We’ll dive into this in another video

Example: Revisiting Bags

Let's revisit the generic "Bag" data structure we saw in our last video

General idea: We create an Object array that stores our "stuff" (type E)

- We can than add things to our bag, search through our bag, and remove stuff from our bag

Are there any limitations of our Bag?

- Can't have an array of type E
    - Need to cast!

Example: Revisiting Bags

BagOfHolding: a Bag with magically enhanced capacity!

- It’s easier to write, and now I can hold *all* my stuff!

Let’s Look at Code

- Bag.java
- BagOfHolding.java

Summary

Vectors are random-access data structures, like an array, but they add new functionality

- Inserting/Removing
- Resizing
- Searching
- Support for “generic” types

The Vector class implements many functions that we will revisit when we discuss the abstract concept of a “List”