Implementing an ArrayList

15-121 Fall 2020

Margaret Reid-Miller

Announcements

- Exam 1: Thursday, Oct 8, 2020
- Drop deadline: Monday, Oct 12
- Grades for Hw3, Quizzes & Labs should be up later this week.
- Homework5: out later today
  - Part I due and Part II checkpoint due Monday, Oct 5 at 11:55 pm
  - Part II complete due Monday Oct 12 at 11:55 pm

Today

- Homework 3: Note, create ONE Scanner object to read from `System.in`
- ArrayList Implementation

ArrayList Implementation

What are 3 ways arrays differ from ArrayLists?

- You can't change the size of an array
- Arrays can hold primitive data values
- Arrays are not instances of a class:
  - We use a special syntax with arrays.
  - Arrays don't have methods.

The basic operations of an array:

• Getting the length of an array:	`a.length`
• Getting an array element:	`a[i]`
• Setting an array element:	`a[i] = …`

All basic array operations run in O(1) time!

We often say arrays are **random access** because you can access or set any element of an array in the same amount of time as the first element. (At least hardware manufacturers try to make that true.)

Analyzing ArrayLists

- We can use an array to implement an ArrayList, just like we did in the contact list application.
- Because arrays underlie ArrayLists, ArrayLists runtime for `size` / `get` / `set` is also O(1).
- But `add` / `remove` at index are O(n) in the worst case.
- Let's see why by implementing an ArrayList.

Recall: When we declare and create an instance of an ArrayList, we specify its type.

type argument

```
ArrayList<Person> contacts = new ArrayList<Person>();
```

Implementing ArrayList

`MyArrayList` is a *generic* class, which specifies a type parameter.

```
public class MyArrayList<E> {
   private E[] values;
   private int size;

   public MyArrayList() {
      values = (E[]) new Object[1];
      size = 0;
   }
```

type parameter

use type parameter as type

no class `E[]` with a constructor

Need to cast to a generic type

Implementing ArrayList `size` & `get`

```
// Returns the number of elements in the list
public int size() {
   return size;
}

// Returns the element at position index
public E get(int index) {
   if (index < 0 || index >= size)
      throw new ArrayIndexOutOfBoundsException(index);

   else
      return values[index];
}
```

constructor

Note: "**`throw`**" is a Java statement, but a method header declares that it "**`throws`**".

Implementing ArrayList `set`

```
// Replaces element at position index with newValue.
// Returns the element previously at position index

public __E__ set(int index, E newValue) {
   if (index < 0 || index >= size)
      throw new ArrayIndexOutOfBoundsException(index);

   _E__ oldvalue = values[index];  // save old
   values[index] = newValue;
   return oldvalue;
}
```

Implementing ArrayList add

```
// Appends obj to the end of the list; returns true
public boolean add(E obj) {
   growIfFull();

   values[size] = obj;
   size++;
   return true;
}
```

Why return a boolean?

Header specified by the Collection Interface; ArrayList is a subclass

Why always return true? When would you return false?

when a Collection does not support duplicates (e.g. Set).

Implementing ArrayList `growIfFull`

```
// Helper method to double the length of values
private void growIfFull(){

   if (size == values.length) {
      E[] bigger = (E[]) new Object[size * 2];

      for (int i = 0; i < values.length; i++) {
         bigger[i] = values[i];
      }
      values = bigger;
   }
}
```

Just as in the constructor

Implementing ArrayList add at index

```
// Insert obj at position index
public void add(int index, E obj) {
   if ( index < 0 || index > size )
      throw new ArrayIndexOutOfBoundsException(index);

   growIfFull();

   for (int from = size-1; from >= index; from--) {
      values[from+1] = values[from];
   }

   values[index] = obj;
   size++;
}
```

Can add at end

`from` is where moving data "from"

Implementing ArrayList add at index (alternate)

```
// Insert obj at position index
public void add(int index, E obj) {
   if (index < 0 || index > size)
      throw new ArrayIndexOutOfBoundsException(index);

   growIfFull();

   for (int to = _size__; to_> index_; to--) {
      values[ to ] = values[_to-1_];
   }

   values[index] = obj;
   size++;
}
```

to is where moving data "to"

Exercises

1. Write the remove method for MyArrayList class:

```
public E remove (int index) {







}
```