Middlebury

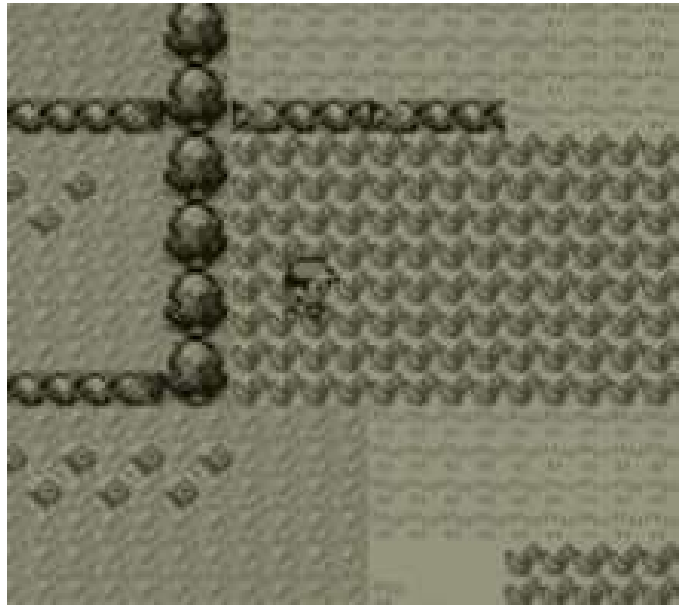
CSCI 201: Data Structures

Spring 2025

Lecture 3W: Java Collections

Goals for today:

- Use `javac` and `java` directly!
- No more new `Java` syntax!
- Identify the difference between an Abstract Data Type and a Data Structure.
- Use an `ArrayList` to store multiple items (of the same type), in which the number of items can increase/decrease based on the needs of your algorithm.
- Use a `HashMap` to store key-value pairs.



First, let's unpack what the VS Code play button is doing.

Open a Terminal in the `lecture06` folder:

```
username@computer$ javac CompileThenRunWithArguments.java
```

```
username@computer$ java CompileThenRunWithArguments
```

Now try:

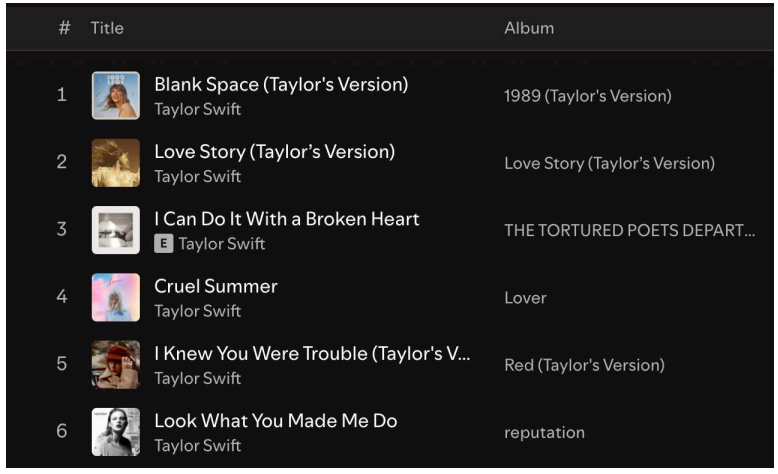
```
username@computer$ java CompileThenRunWithArguments x y c 123 MikeWazowski
```

```
public class CompileThenRunWithArguments {
  public static void main(String[] args) {
    System.out.println("Here's what was passed to the program:");
    for (int i = 0; i < args.length; i++) {
      System.out.println("Argument [" + i + "]: " + args[i]);
    }
    if (args.length == 0) System.out.println("Nothing!");
  }
}
```

Motivating dynamic storage

How would you keep track of which Pokémon a player has?

How would you organize songs in a music playlist?



These tasks would be a bit hard to do with fixed-size arrays (directly).

Motivating dynamic storage

Imagine we had a utility to keep track of this - what methods would you like?

add(item): append item to end of list

remove(index): remove item at specific index

get(index): retrieve item at index

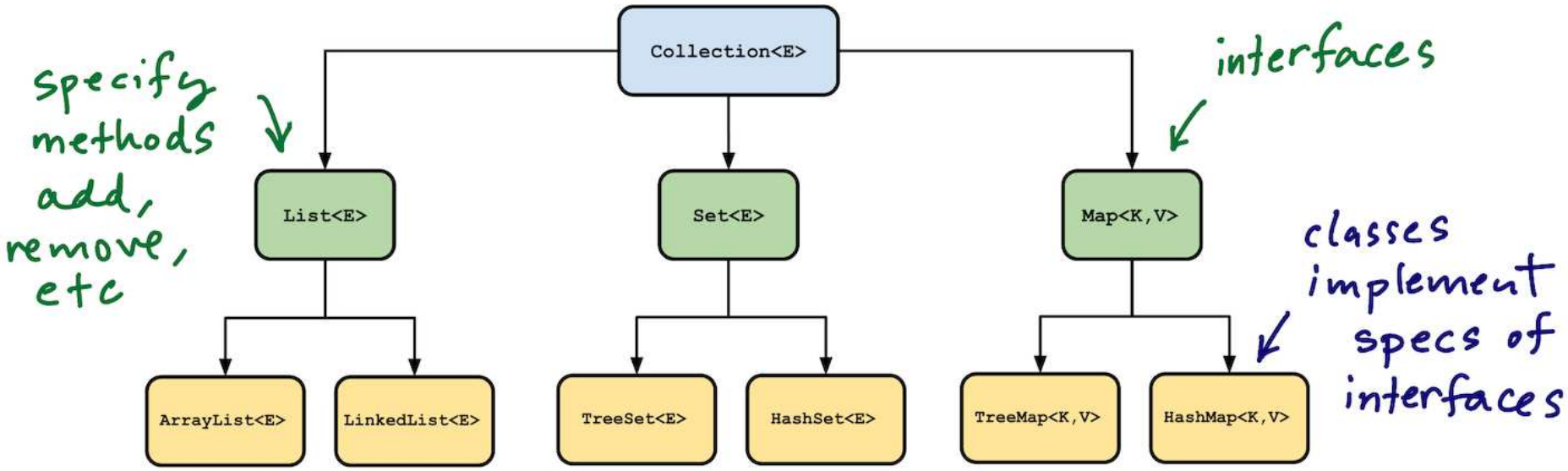
indexOf(item): find index of item

size(): # of items stored

sort(): sort items

There are tools (built into Java) to help with this.

A collection represents a group of objects, known as its elements.



- **Abstract Data Type (ADT):** formal description of behavior of data type, e.g. a `List` allows accessing item at a specified index (but implementation can vary).
- **Data Structure:** concrete organization/representation of data (implements spec defined by an ADT). Example: `ArrayList`.

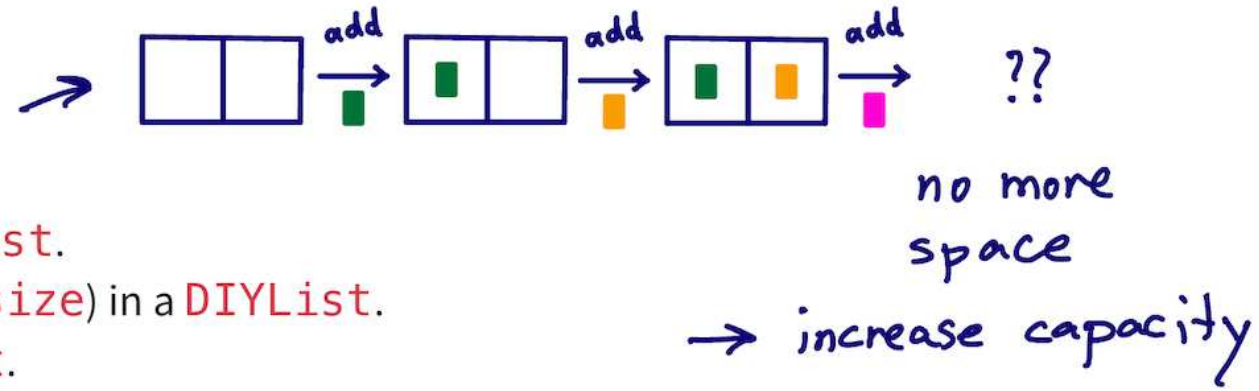
Okay, let's take a shot at a `List` ourselves.

We'll design our own implementation of a `List` called `DIYList`.

But first, we should check the `List` spec:
https://docs.oracle.com/javase/8/docs/api/java/util/List.html

Let's focus on methods to:

1. Construct an empty `DIYList`.
2. `add` an item to a `DIYList`.
3. `remove` an item from a `DIYList`.
4. Retrieve the number of items (`size`) in a `DIYList`.
5. `clear` the items in a `DIYList`.



Idea: use a fixed-size array that has enough space (capacity) to hold our items.

If we need more space, just allocate a new larger array and copy items!

Adding (add) an item to a DIYList.

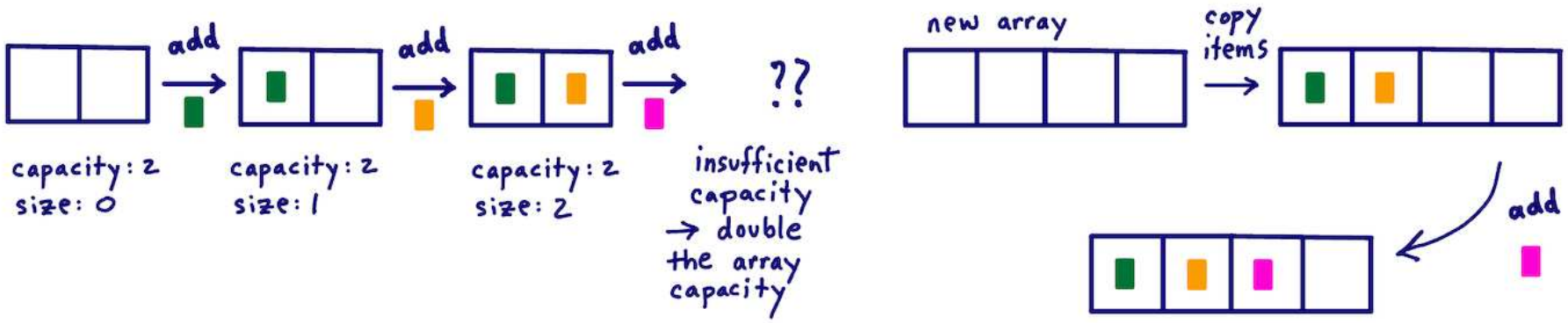
Without loss of generality, imagine our DIYList can only hold String items.

```
public class DIYList {
  int size; // current number of items actually stored
  String[] items; // capacity is items.length

  ...

  public void add(String item) {
    // Is there enough space (capacity, i.e. items.length)?
    // If not, make more space and copy the old items.

    // Place item in items[size] and increment size.
  }
}
```



Removing (`remove`) an item from a `DIYList`.

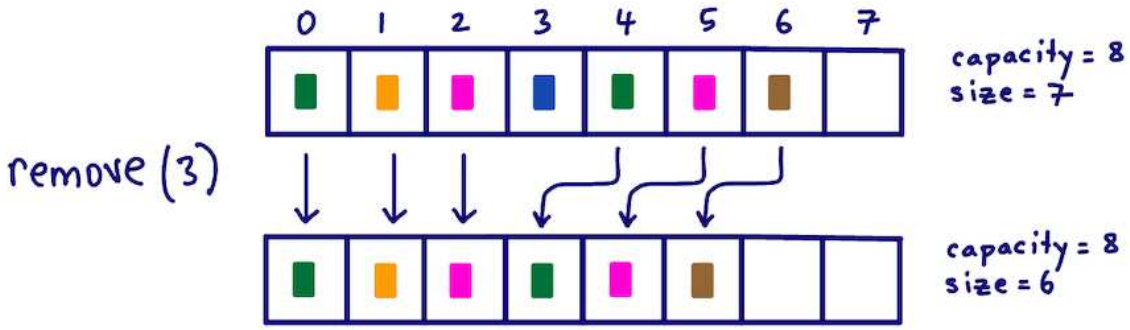
```
public class DIYList {
  int size; // current number of items actually stored
  String[] items; // capacity is items.length

  ...

  public void remove(int index) {
    // if index is beyond the index of the last item, nothing to do -> return

    // replace the item at index with the item at index + 1
    // replace the item at index + 1 with the item at index + 2
    // replace the item at index + 2 with the item at index + 3
    // ... until all items after the index have been shifted left

    // decrement size (because we removed an item)
  }
}
```



Use a for loop, starting at index of item to be removed, until size.
In each iteration set items[i] = items[i+1]

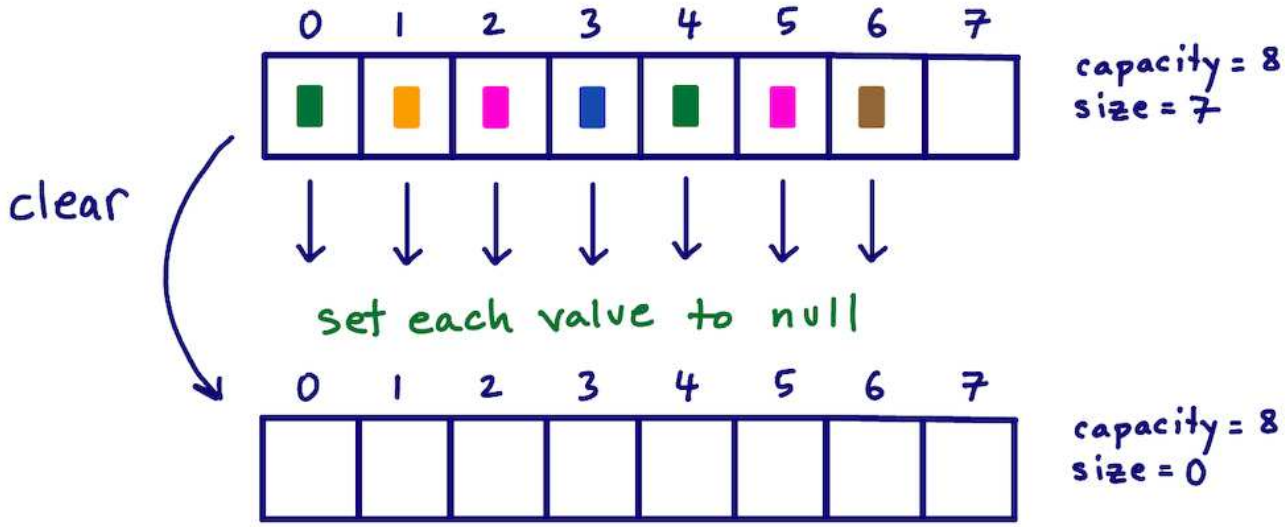
Clearing (`clear`) the items in a `DIYList`.

```
public class DIYList {
  int size; // current number of items actually stored
  String[] items; // capacity is items.length

  ...

  public void clear() {
    // set all items to null

    // set size to 0
  }
}
```



The truth is that we just implemented an `ArrayList`!

Main idea of an `ArrayList`:

- Internally use an array to hold the items.
- This array needs to have enough space (capacity) to hold the items.
- To add an item:
  - First check if there is enough space.
    If not, make a new array with more space and copy the old items into this array.
  - Add the new item to the next empty slot.
- How should we increase the capacity?

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#ensureCapacity-int-

Each `ArrayList` instance has a *capacity*. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an `ArrayList`, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost.

An application can increase the capacity of an `ArrayList` instance before adding a large number of elements using the `ensureCapacity` operation. This may reduce the amount of incremental reallocation.

Let's practice with `ArrayList`s.

Note that `Java Collection`s only work with reference types
We cannot directly use primitive types.

How do we use them with `int`, `double`, `char` etc.?
Luckily, there are wrapper classes called `Integer`, `Double`, `Character`, etc.

```
import java.util.ArrayList; // don't forget this!
// import java.util.*; // can also be convenient to import everything in java.util

public class ArrayListExamples {
  public static void main(String[] args) {
    // initialize empty array list
    ArrayList<Integer> list = new ArrayList<>();

    list.add(2);
    list.add(12);
    list.add(5);
    list.add(6);

    System.out.println("list size = " + list.size()); // 4

    for (int i = 0; i < list.size(); i++) {
      System.out.println("Item[" + i + "]: " + list.get(i));
    }

    list.remove(2); // remove item at index 2 (i.e. the 5)
    System.out.println("list size = " + list.size()); // 3
  }
}
```

`HashMap`: another useful collection to know about.

- `HashMap<K, V>`: stores key-value pairs (keys have type `K` and values have type `V`).
- **Useful methods:** `containsKey` (checks if a key of type `K` exists), `put` (inserts a key-value pair), `get` (retrieves the value associated with some key), `keySet` (retrieves a `Set` of all keys).

```
1  import java.util.HashMap;
2  // import java.util.*; // <-- this can be more convenient!
3
4  public class HashMapExample {
5    public static void main(String[] args) {
6      String lyric = "and i love vermont, but it's the season of the sticks";
7
8      HashMap<Character, Integer> frequency = new HashMap<>();
9      for (int i = 0; i < lyric.length(); i++) {
10       char c = lyric.charAt(i);
11       if (frequency.containsKey(c)) {
12         frequency.put(c, frequency.get(c) + 1);
13       } else {
14         frequency.put(c, 0);
15       }
16     }
17
18     Set<Character> characters = frequency.keySet();
19     for (Character c : characters) {
20       System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21     }
22   }
23 }
```

Find The Bug! 

Here it is! 

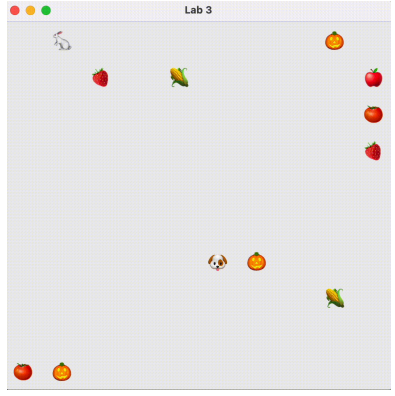
```java
import java.util.HashMap;
// import java.util.*; // <-- this can be more convenient!

public class HashMapExample {
  public static void main(String[] args) {
    String lyric = "and i love vermont, but it's the season of the sticks";

    HashMap<Character, Integer> frequency = new HashMap<>();
    for (int i = 0; i < lyric.length(); i++) {
      char c = lyric.charAt(i);
      if (frequency.containsKey(c)) {
        frequency.put(c, frequency.get(c) + 1);
      } else {
        frequency.put(c, 1); // first time we encounter a character, it counts, so insert 1 not 0
      }
    }

    Set<Character> characters = frequency.keySet();
    for (Character c : characters) {
      System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
    }
  }
}
```

See you Friday!



- We'll practice using `ArrayList`s in Friday's lab.
- `HashMap`s were just introduced now in case you find them helpful to solve problems (we'll talk about the underlying data structures later in the semester).
- `HashSet`s can also be useful if you want to store an unordered set of items.
- Read the write-ups and download the starter code for Lab 3 and Homework 3 in preparation for lab on Friday.
- Reminder that Noah (go/noah) and Smith (go/smith) have office hours throughout the week and the 201 Course Assistants have drop-in hours in the late afternoons/evenings (go/cshelp).