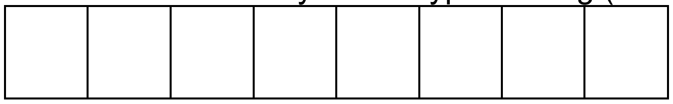
Lecture 1 (6/24)

- Introduction: course website - your go-to for all information!
    - Syllabus
        - Office hours (except this week)
        - Discussion section
        - Grading scheme
        - Exams - closed book
        - Late days
        - Cheating policy
    - Working at home (jGrasp)
    - IPL
    - Message board
- Overview of the course
    - CSE 142
        - Control structures
            - methods: params/returns
            - if/else
            - for loops/while loops
        - Data structures
            - int/double/char... (primitives)
            - Files, Scanners
            - arrays
            - Objects (Critters - state and behavior - the focus of this course)
    - CSE 143
        - More data structures
            - ArrayList, LinkedList
            - Binary trees
        - More control structures
            - Recursion
        - OOP
            - Interfaces
            - Inhereritance
    - Central theme: client view vs. implementation view
        - Who knows how to ***use*** a radio?
        - Who knows how to ***build*** a radio from parts at RadioShack?
        - Client view: knowing ***what*** object is, how to use it
        - Implementation view: knowing ***how*** it works
        - We'll be switching back and forth, which makes it complicated
- Array review
    - Basics - can store many of one type of thing (a whole bunch of buckets)



        - Type (what kind of things are stored)
        - Elements (the things being stored)
        - Length (the number of things that can be stored)

        - Index (a location in the array)
    - Operations
        - To create: int[] arr = new int[8];
        - To get: int x = arr[3];
        - To set: arr[4] = 10;
        - For the length: arr.length
    - Code: read lines from data.txt into an array

```
String[] lines = new String[1000];
Scanner input = new Scanner(new File("data.txt"));
int lineCount = 0;
while (input.hasNextLine()) {
   String line = input.nextLine();
   lines[lineCount] = line;
   lineCount++;
}
String firstLine = lines[0];
String lastLine = lines[lines.length - 1];
for (int i = lines.length - 1; i >= 0; i--) {
   System.out.println(lines[i]);
}
```

    - What's wrong with the previous code?
        - Fixed size - will print out lots of nulls at the end
        - Could fix by changing lineCount for lines.length
        - But a better solution: ArrayList
- ArrayList
    - Fits our idea of a list: can add something, can remove things, *can change size*
        - Probably the **most commonly used** data structure in Java
    - Starts out empty, you can add things to it, keeps track of the order
    - When you create a new ArrayList, you have to tell Java what type of thing you're putting in it
        - **"Generics"** - new, allows lists to store different types of things
    - Translation:

```
// translation from array to ArrayList:
//    String[]          => ArrayList<String>
//    new String[10]    => new ArrayList<String>()
//    a.length          => list.size()
//    a[i]              => list.get(i)
//    a[i] = value;     => list.set(i, value);
// new operations:
//     list.remove(i);      --remove the ith value
//     list.add(value);     --appends a value
//     list.add(i, value); --adds at an index
//     list.clear()         --remove all value
//     list.toString();     --nice String of the list
```

    - Guide: the Java API
        - **"Collections framework"** - a bunch of really good tools (we'll discuss)
    - Rewrite code with ArrayList:

```
ArrayList<String> lines = new ArrayList<String>();
Scanner input = new Scanner(new File("data.txt"));
while (input.hasNextLine()) {
   String line = input.nextLine();
   lines.add(line);
}
String firstLine = lines.get(0);
String lastLine = lines.get(lines.size() - 1);
for (int i = lines.size() - 1; i >= 0; i--) {
   System.out.println(lines.get(i));
}
```

- You can also write code with integers, for example:
  - But must use WRAPPER class

    ```
    ArrayList<Integer> list2 = new
    ArrayList<Integer>();
    list2.add(42);
    list2.add(3);
    list2.add(-1);
    list2.add(0);
    list2.add(101);
    int first = list2.get(0);
    int numElements = list2.size();
    System.out.println("list2 = " + list2);
    ```

- Implementing ArrayList
  - We just talked about the client view - how to use ArrayList
  - Let's look inside it - **implementation view**
    - Lets us talk about how to design structures
    - Useful for general programming of objects/classes
    - **"Software cadaver"** - just like med students dissect cadavers, we're dissecting software
  - If I showed you Java's ArrayList now, you'd all drop the course
    - It's scary, so we'll start simpler - design our own ArrayIntList class
    - Only stores int values
    - But still not simple enough - we'll develop the code in stages
    - In the end - something that closely resembles ArrayList
  - First, we need to know how to USE an ArrayList, so we know what kinds of things we'll need
    - ArrayIntListClient
    - Let's have it do some basic operations
    - As the name says, we're going to implement it with an array
    - What do we need to represent the data?
      - Need the array
      - Need the size
      - But we'd need 2 arrays and 2 sizes to represent the two lists
  - How can we do it?
    - Same idea as with the "data.txt" example, but ENCAPSULATED
    - Unfilled array

- Let’s create a new ArrayIntList class
    - The array and the size become our FIELDS
    - elementData --> parallels Java’s version
    - Not like local variables - they are the STATE or innards of the object, one set per instance of the object
        - e.g. each radio has its own circuitry inside, each car has its own steering wheel
        - Stay around indefinitely - don’t “disappear” when they go out of scope
        - So now instead of 4 variables, we have 2 objects
- (ADD CONSTRUCTOR CALLS)
    - Use debugger to show what happens - each has its own elementData and size
    - But elementData is set to NULL - special 0-equivalent meaning “no value”
        - We never told it to construct a new array!
    - We could initialize it when declared, but this is BAD
        - Job of the constructor, a special method called when you create a new object using **new**
    - Constructor has special syntax - no return type, same name as the clasa
        - This is how Java knows to call it when you say **new**