CS61B Lec 1

Sean Villegas
Prof. Hug

May 25, 2025

Defining a Typical Class Terminology

- If a method is going to be invoked by an instance of the class, then it must be non-static (i.e. remove the **static** to make it **public void**)
- There are no de-structors in java, the garbage collector handles it. C++ will use that for memory management
- You put the world new almost every time you make a new object
- When we compare the `Dog largerDog = Dog.maxDog(maya, hugeGreg);`, we return the same object, instead of a copy. Vs. python mutability that returns a copy
- **Moral:** you do not need to specify `this.<var>` if there is no naming conflict, meaning Java can infer with just the **var** name

```
public Dog maxDog(Dog otherDog) {
    if (this.size > otherDog.size) { // you do not need to specify //
        return this;
    }
    return otherDog;
}
```

- Never have a static variable that changes in a class. It is dangerous. Prof advises against using static variables in general
- `final` in java is a constant keyword. After a variable has been initialized and declared as final it cannot be changed

Lists in Java

- List is a universal concept of ordered objects
- Lists are abstract; significado:
- Expect to use `import java.util.List;`, then `import java.util.<dataType>List;` `List varName = new <dataType>List();` initializer

- `add` for appending
- `ArrayList` is the most common but sometimes you will see `LinkedList`. The main idea is that there are multiple techniques under the hood to get the same effect
- Specifying lists can be a little lower level because we have to specify.
- The python list is also an array list
- `LinkedList` is faster at computation in popping the front item vs. `ArrayList`
- `ArrayList` is faster for popping the last element
- **Abstraction in Lists means there is a contract for what a list should do without dictating how it should do it.** Thus any operations from the List library satisfy the List contract, just with different internal mechanisms.
- `LinkedList` uses doubly linked list
- `ArrayList` uses dynamic arrays, *dynamic* meaning to automatically resize itself as elements are added or removed compared to traditional arrays that have a set size of memory
- You need to specify the type of the `List`. When the type is known, the compiler can perform optimizations that might not be possible with a raw, untyped list, and you avoid runtime errors.
- In the demo, Prof uses raw use of parameterized class 'List', and it works fine. But Intellij is mad

```
public class ListJ {
    public static void main(String[] args) {
        List L = new LinkedList();
        L.add("a");
        L.add("b");
        // System.out.println(L.get(0)); // works
        // String s = L.get(0); // doesn't work, you must do List<Str
    }
}
```

- In Intellij you can write `Object s = L.get(0);` and it will replace it based on the object
- Settings constraints in Java is the focus of CS61B

Arrays

- **array** is a restricted version of the list ADT

- Newly learned: Has no methods
- So why does Java have lists and arrays?
  - **Because arrays are faster, performative, less memory**
- Why does Java favor arrays over lists?
  - Lists are built on top of the array
  - Java is built for performance over elegant/simple code

Maps

- Key-value pairs. Each key is trusted to be unique.
- Symbol table, Associative Array (theoretical cs), and Dictionary in Python
- Maps

```
import java.util.Map;
import java.util.TreeMap;

public class Maps {
    public static void main(String[] args) {
        Map<dataType, dataType> Demo = new typeMap<>();
            Demo.put("key", "value");
            Demo.put("key1", "value");
            String output = Demo.get("key1");
            System.out.println(output);
    }
}
```

- Hash Table v.s. TreeMap discussed differently, most commonly used. Used in python as well.
- You must specify both types in **maps**

Course Logistics

- Fully understand linked lists and array lists; covered in Midterm 1. Takes 3 lectures to understand