Abstraction

CSCI 136 :: Williams College

This Video

- Abstraction
  - What & why
- Exploring interfaces & abstract classes
  - We’ve used them, but we haven’t dug into the details
- Look at the Structure5 Hierarchy

Abstraction is Beautiful

- Abstraction lets us solve complex problems elegantly by ignoring the "irrelevant" details
  - What does it mean to be irrelevant?
    - As a systems researcher, I spend a lot of time on the “irrelevant details”, but that is an even stronger argument in favor of abstraction…
  - What does it mean for a problem to be complex?
    - “Quick script” vs. a “program”

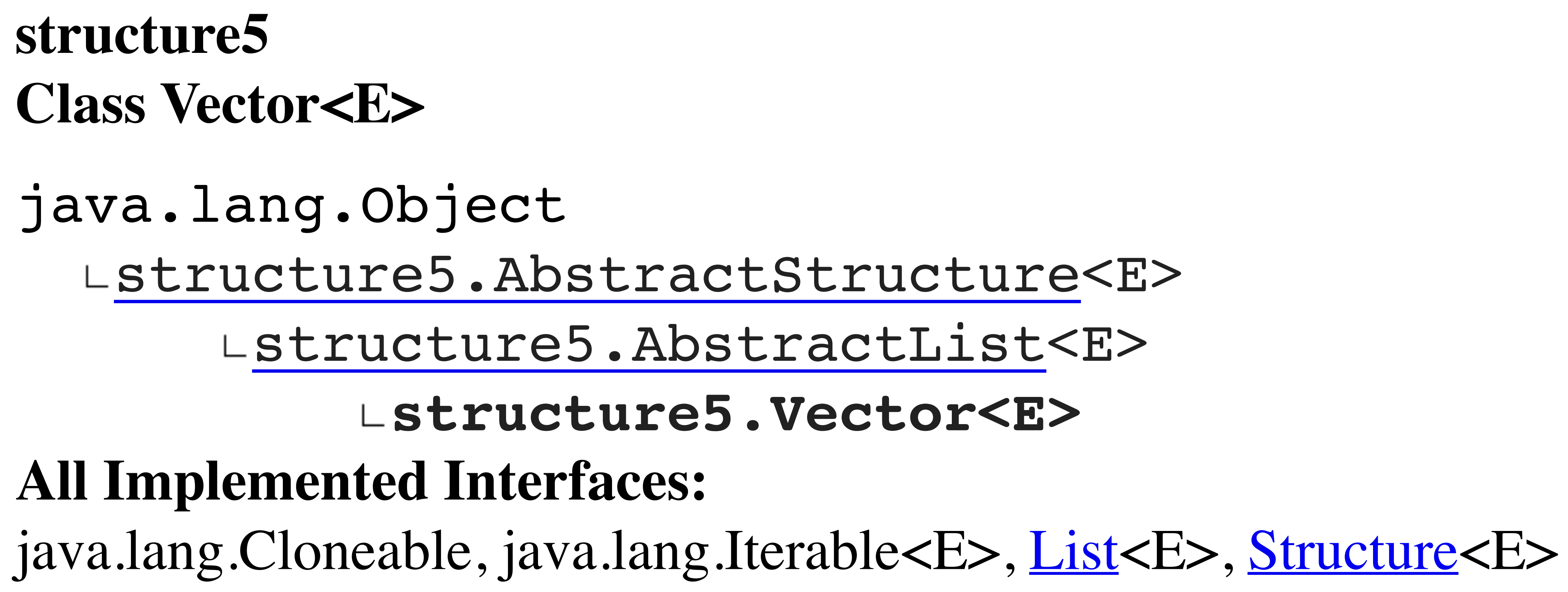
As humans, we simply can't reason about complex systems without breaking the problem down into reasonably-sized, simplistic parts.

We Already Use Abstraction

- How have we seen abstraction so far in CS136?
  - We started using `Vector` objects before we looked at how they were implemented. How is that possible?
    - We learned the function behaviors (inputs + outputs) before we learned the data structure implementation (member variables, method code)
  - We used `public/private/protected` to help us to hide implementation details

We Already Use Abstraction

- We've also benefited from abstraction without explicitly saying so
- Vector extends and implements other Java classes/interfaces



structure5
Class Vector<E>

```
java.lang.Object
  └structure5.AbstractStructure<E>
      └structure5.AbstractList<E>
          └structure5.Vector<E>
```

All Implemented Interfaces:
java.lang.Cloneable, java.lang.Iterable<E>, List<E>, Structure<E>

Java gives us two very powerful tools for abstraction:
the Interface and the Abstract class

Abstraction helps us to be Lazy

- We often optimize algorithm performance by minimizing big-O
- But once I heard how much some engineers get paid*, I started to appreciate other optimization targets: saving programmer’s time

- Let's figure out how to save the programmer’s time in two ways:
  - Code that *uses* data structures should be faster to write
  - Code that *implements* data structures should be faster to write

Saving programmer effort: Interfaces Define *Behavior*

- Consider the List interface:
  - How many programs have we looked at that use classes that implement the `List` interface?
  - Do we care which class is used as long as it implements `List`?
    - MAYBE!
    - But we can write our code in ways that let us pick a specific class later

An Interface defines a Contract

- If a class implements an interface, it must adhere to that contract
  - This means the class must implement *all* methods in the interface
  - But as a result, we can swap any class that implements the interface into this sample code in place of `SinglyLinkedList`:

```
public static void main(String[] arguments)
 {
     List<String> argList = new SinglyLinkedList<String>();
     for (int i = 0; i < arguments.length; i++){
         if (!argList.contains(arguments[i])){
             argList.add(arguments[i]);
         }
    }
    System.out.println(argList);
 }
```

Takeaway: an interface defines behaviors, and that is all a programmer needs to start writing functional code

Saving programmer effort: Inheritance allows reuse

- Are there `List` methods that we can write without knowing the low-level implementation details?
  - Let's look at the `AbstractList` class
    - Are there methods with real code?
      - Yes
    - Are all of the methods in the `List` interface present?
      - No. Otherwise it wouldn't be *abstract*

Saving programmer effort: Inheritance allows reuse

- A programmer can *extend* an (abstract) class and complete its implementation
  - This makes the class ***concrete***.
- Lets look closely at the code for the `SinglyLinkedList` class
  - It overrides some `AbstractList` methods with its own implementations
  - It entirely omits implementations for others

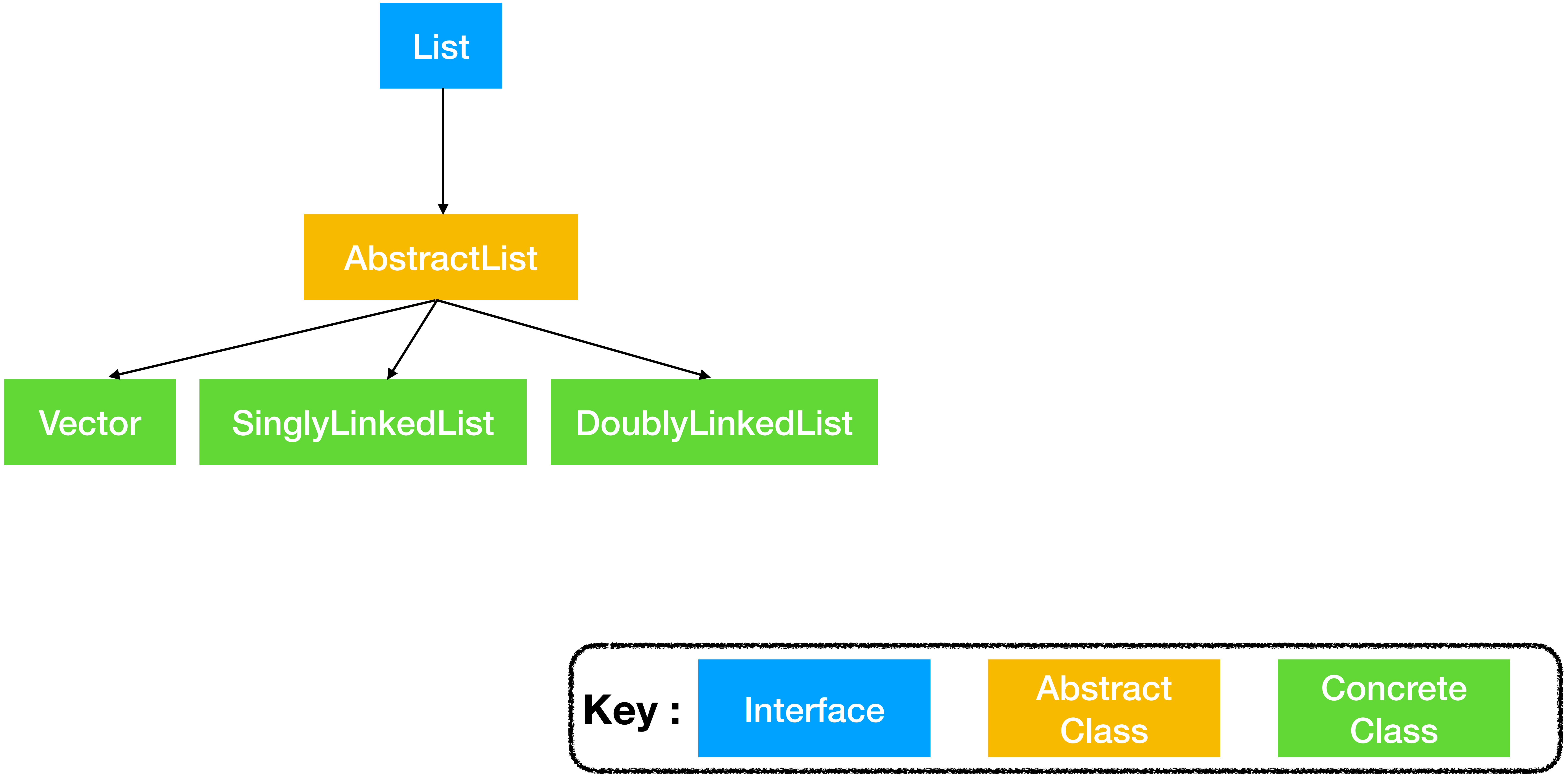
Takeaway: an abstract class defines behaviors AND it lets us define general code. We *can* overwrite that code as needed.

One Last Note

- If an abstract class is like an interface but gives us the added flexibility to provide code, why have interfaces at all?

A class can extend at most one class but implement any number of interfaces.

Structure5 Hierarchy (So Far…)



List

AbstractList

Vector

SinglyLinkedList

DoublyLinkedList

Key :

Interface

Abstract Class

Concrete Class

Review of Java Tools for Abstraction

- `public/private/protected`
  - Visibility modifiers let us “hide” a class’s low-level details
    - Maintain control over variable access to prevent illegal program states
    - A program that only uses public methods doesn’t need to change when we change our class’s implementation
- Interfaces
  - Define a ‘contract’ so we can write implementation-agnostic code
- Abstract Classes
  - Specify behavior & let us provide partial implementation