## Abstraction

## Object oriented programming

- One advantage: Helps break down data and code into self-contained chunks

- Also: can use objects as building blocks to create other objects!
  - Improved portability, extensibility
  - *Avoid repetition!*

- Today: abstract classes

## Recall: Interfaces

- “Recipe” for the methods that must be available for any class implementing the interface
- Allows us to use multiple objects of different class types, through a united interface

- Example: TwoPlayerGame.java
  - We could run any game with the same code because they all used the same interface!

## Limitations of interfaces

- Can’t write any code in an interface
  - (for now)

- When is that a problem?

## Limitations of interfaces

- Can’t write any code in an interface
  - (for now)

- When is that a problem?

- What if several different classes implement the same exact method?
  - Same signature
  - AND same exact code

## Example: Lists

- Vector and SinglyLinkedList both implement the List interface
- That means they both have a method addFirst(E)

## Example: Lists

- Vector and SinglyLinkedList both implement the List interface
- That means they both have a method addFirst(E)
- In both cases, the method is as follows:

```
public void addFirst(E value) {
        add(0,value);
 }
```

## Abstract Classes

- Goal: if similar classes have *identical* methods, just write that method once
- For each class, tell Java “I want to use that method I already wrote.”

- Tool: create an *abstract class* to store these methods

## Abstract class: definition

- An abstract class is a *partial* implementation of a class; uses `abstract` keyword
- Has some methods written out
  - Can also have instance variables
- Don't need all methods, even if implementing an interface
- Like an interface, cannot *instantiate* an object of an abstract class type

- Idea: this is just a part of a class! Need to fill in details with a normal (not abstract) class

## Usage

- We write some methods in an abstract class
- Then, our other classes use the `extends` keyword to tell Java that they are using this abstract class

- If we extend an abstract class, we get to use all of its methods!  Plus any we implement

## Back to Lists

- In structure5, have an AbstractList class that implements methods that would be identical in all Lists
  - addFirst, addLast, contains, etc.

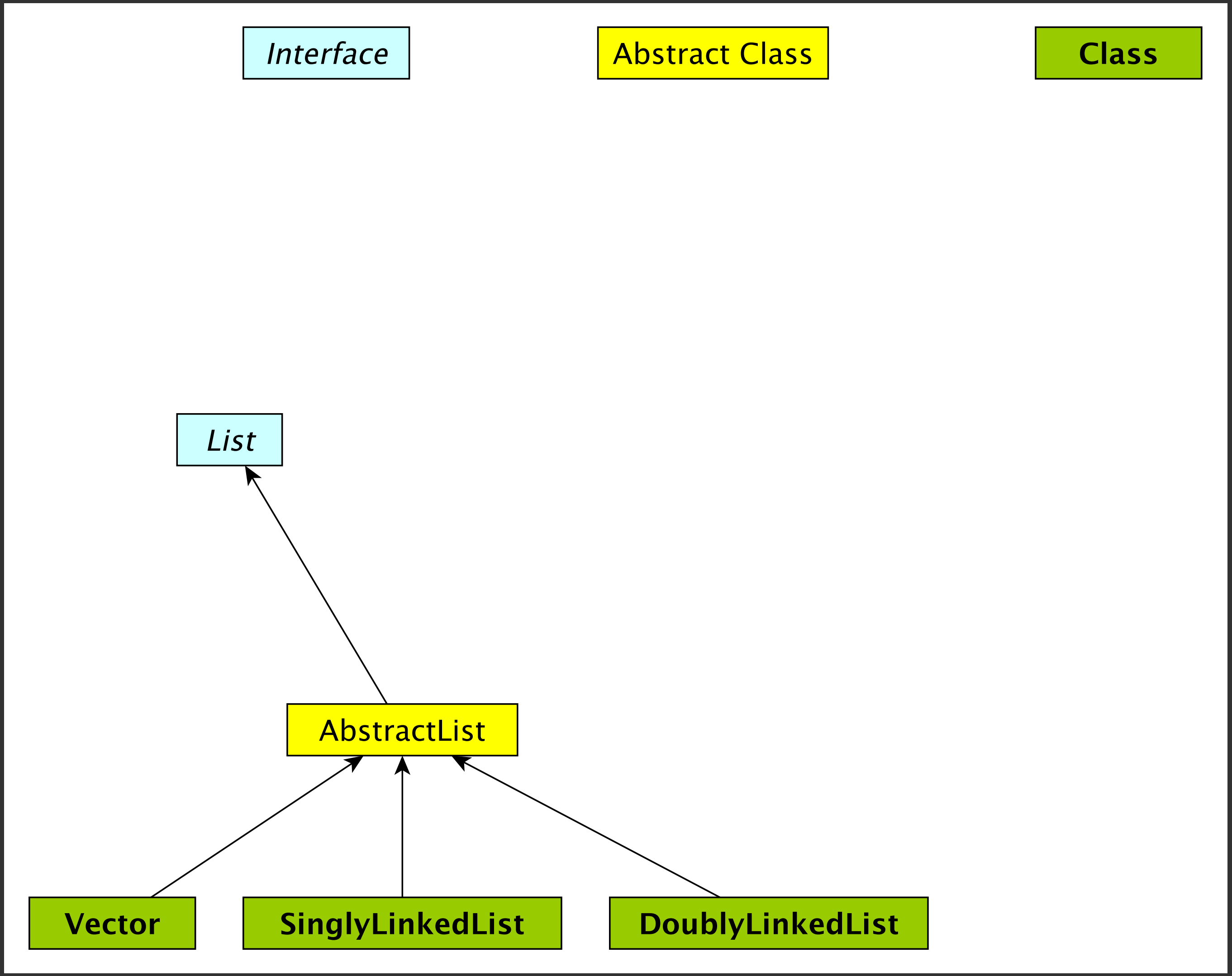
- Our lists then extend AbstractList to allow us to use these methods

- Let’s look at the code

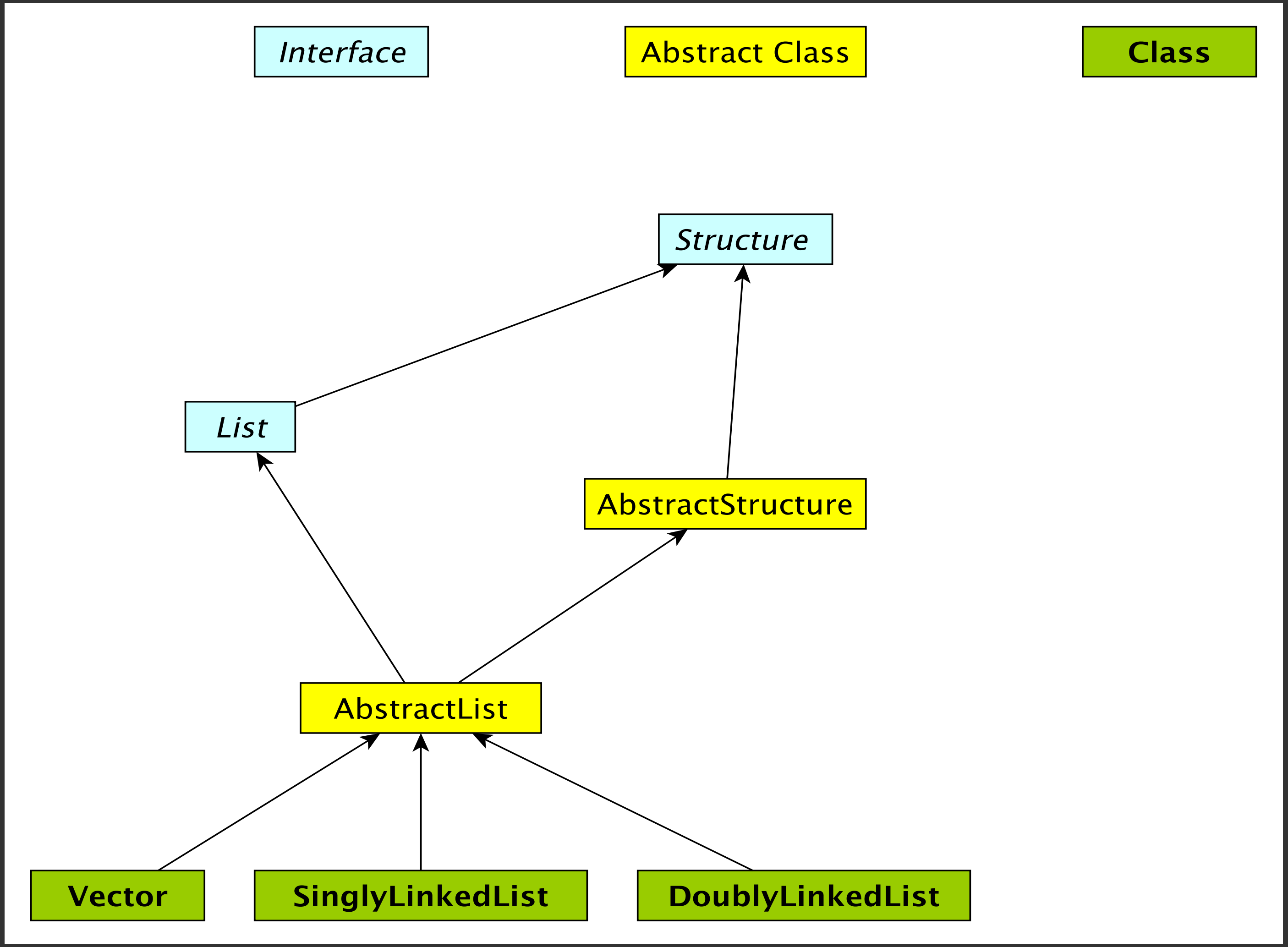
## Summary

- `abstract` keyword declares a class as abstract
- `extends` means that we are adding more methods on to an existing abstract class
- We can replace abstract class methods with our own if we want, or use them as-is
- Can only instantiate concrete (not abstract) classes

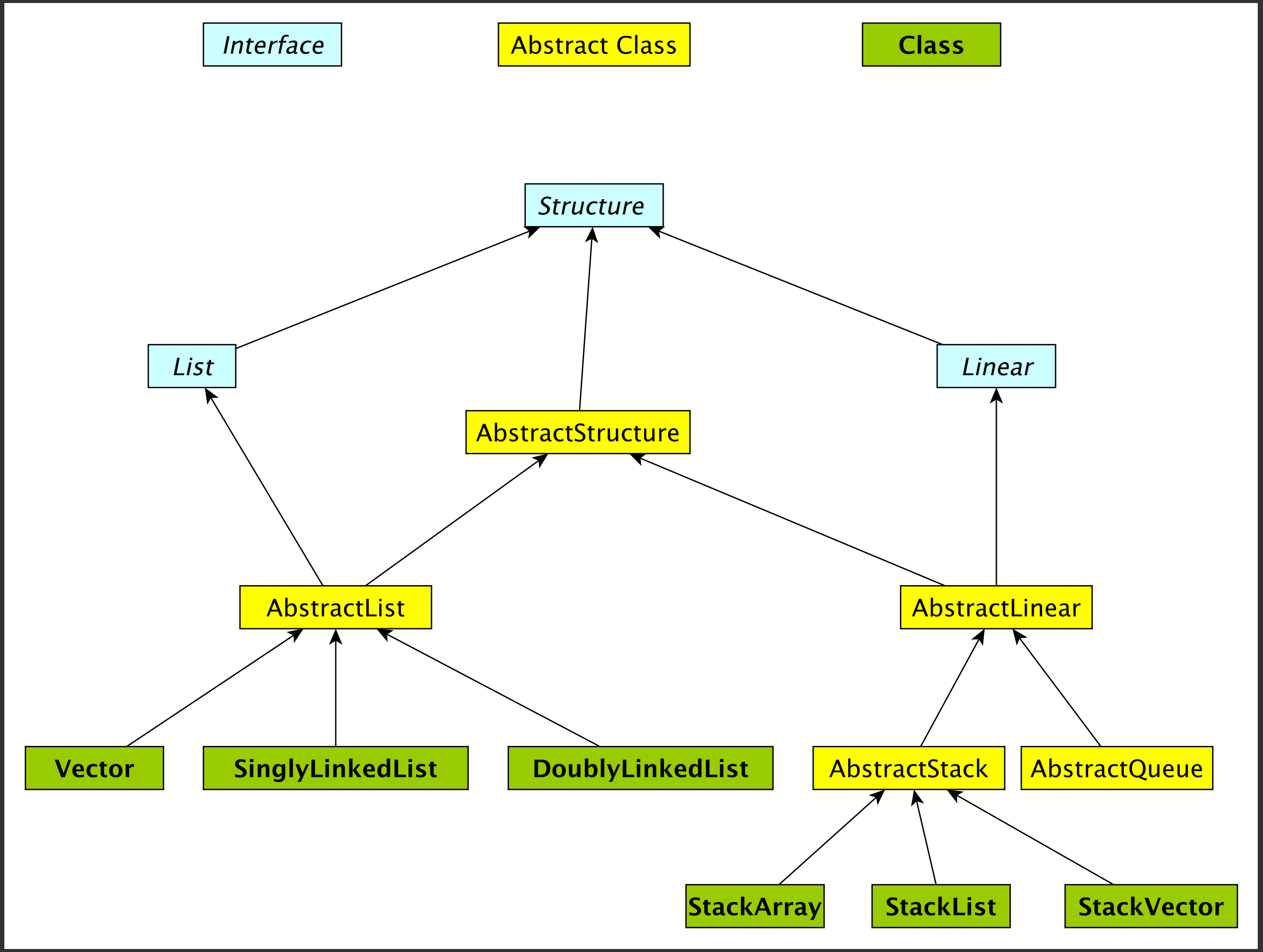
# The Structure5 Universe (almost)



# The Structure5 Universe (so far)



# The Structure5 Universe (soon)


Interface
Abstract Class
Class
Structure
List
Linear
AbstractStructure
AbstractList
AbstractLinear
Vector
SinglyLinkedList
DoublyLinkedList
AbstractStack
AbstractQueue
StackArray
StackList
StackVector