CSCI 136:
Data Structures and
Advanced Programming
Lecture 12
Abstract data types
Instructor: Dan Barowy Williams

Topics
• ADTs
• More linked lists

Your to-dos
1. Read before Mon: Bailey, Ch 6-6.3.
2. Lab 4 (partner lab), due Tuesday 3/9 by 10pm.

Announcements
• **Midterm exam**, in lab, Thursday, March 17.
• **Friday colloquium**, Elena Glassman (Harvard), 2:35pm in Wege Auditorium

“Human-AI (Mis)Communication: challenges and tools for successfully communicating what we want to computers”

Abstract
While we don’t always use words, communicating what we want to a computer, especially an artificially intelligent one, is a conversation—with ourselves as well as with it, a recursive loop with optional steps depending on the complexity of the situation and our request. I will present some key, perhaps previously under-appreciated steps and describe conditions where it is critical to support them, illustrated with examples from recent publications on (1) novel interfaces for interactive program synthesis and (2) interactive visualizations of large piles of complex data. In the process, I will describe relevant theories from the learning sciences, i.e., Variation Theory and Analogical Learning Theory, that have design implications for future interface and interactive system design—to hopefully maximize the bidirectional speed and accuracy of human-AI communication.
Quiz

The purpose of a class:
To “abstract away” implementation details.

Abstraction

Abstraction is the process of removing irrelevant information so that a program is easier to understand.

Do you see any similarities?

The two classes share the same interface.
Interface

An interface defines boundary between two systems across which they share information. An interface is a contract: calling a method defined in an interface returns the data as promised.

Because an interface contains no implementation, programmers who use them cannot rely on implementation details.

E.g., the List interface states that there must be an add method but does not say how it should be implemented.

List

A list is an ordered collection of items of an element of type E. It supports prepending an element to the front, appending (adding) and element to the end, finding an element, and element removal.

A Vector is a list.

A SinglyLinkedList is a list.

A DoublyLinkedList is a list.

Observe that this similarity is “deeper” than just what an interface provides....

Abstract Data Type

An abstract data type is a mathematical formulation of a data type. ADTs abstract away accidental properties of data structures (e.g., implementation details, programming language). Instead, ADTs contain only essential properties and are concisely defined by their logical behavior over a set of values and a set of operations.

In an ADT, precisely how data is represented on a computer does not matter.

By contrast: data structure

A data structure is the physical form of a data type, i.e., it is an implementation of an ADT. Generally, data structures are designed to efficiently support the logical operations described by the ADT.

For data structures, precisely how data is represented on a computer matters a lot. Simple data structures are often composed of simple representations, like primitives, while more complex data structures are composed of other data structures.

Vector, SinglyLinkedList, etc. are data structures.
A Vector is a List

```java
structure:
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>

public class Vector<E>
extends AbstractList<E>
implements java.lang.Cloneable

operation                      worst       best
add(int i, E e)                O(n)        O(1)
get(int i)                      O(1)        O(1)
indexOf(E e)                   O(n)        O(1)
remove(E e)                    O(n)        O(1)
size()                         O(1)        O(1)
```

A Linked List is a List

```java
structure:
Class SinglyLinkedList<E>
java.lang.Object
structure5.AbstractStructure<E>
structure5.AbstractList<E>
structure5.SinglyLinkedList<E>

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

public class SinglyLinkedList<E>
extends AbstractList<E>

operation                      worst       best
add(int i, E e)                O(n)        O(1)
get(int i)                      O(n)        O(1)
indexOf(E e)                   O(n)        O(1)
remove(E e)                    O(n)        O(1)
size()                         O(n) [O(1) w/mod.] O(n)
```
ADTs cannot be expressed in Java

At least not directly.

Instead, Java uses types to stand in for ADTs.

Because types in Java are often bound to an implementation, Java provides two mechanisms for programmers to specify a type with varying degrees on a mechanism: interfaces and abstract classes.

Missing from Java: ADT behavior

Java provides no way of specifying behavior independently of implementation.

E.g., a List interface might require

public void prepend(T elem)

But there’s no way to require that the implementation actually place the element at the beginning of the list.

Interface

An interface defines boundary between two systems across which they share information. An interface is a contract: calling a method defined in an interface returns the data as promised.

An interface contains no implementation!

You cannot specify behavior at all!

Honkable
Abstract class

An abstract class is a partial implementation, mainly used as a labor-saving device.

E.g., many List implementations will implement methods the same way. Why duplicate all that work?

isEmpty() can always be implemented by checking that size() == 0.

"We will encourage you to develop the three great virtues of a programmer: laziness, impatience, and hubris."

—Larry Wall, inventor of the Perl programming language

AbstractHonkable

Laziness. The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful, and document what you wrote so you don’t have to answer so many questions about it. Hence, the first great virtue of a programmer.
Inheritance
(cf. laziness)

*Inheritance* is a *mechanism* for defining a class in terms of another class. It is a labor-saving device employed to reduce *code duplication*. Inheritance allows programmers to specify a new implementation while:

1. maintaining the same behavior,
2. reusing code, and
3. extending the functionality of existing software.

Recap & Next Class

**Today:**

- ADTs
- Lists

**Next class:**

- Sorting