Lecture 1: Intro to ADTs

CSE 373: Data Structures and Algorithms

Course Overview

Course Goals

- Design data structures and algorithms by implementing and maintaining invariants.
- Analyze the runtime and design values of data structures and algorithms
- Critique the application of data structures and algorithms towards complex problems
- Prepare for technical interviews

Course Topics

- Data Structures and ADTs: lists, stacks, queues, sets, dictionaries, arrays, linked lists, trees, hash tables, priority queues, binary heaps and disjoint sets
- Algorithm analysis: Big O Notation, asymptotic analysis, P and NP complexity, Dynamic Programming optimization
- Sorting algorithms: selection, insertion, merge, quick, more…
- Graphs and graph algorithms: graph search, shortest path, minimum spanning trees

Basic Definitions

Data Structure

- A way of organizing and storing data
- Examples from CSE 12X: arrays, linked lists, stacks, queues, trees

Algorithm

- A series of precise instructions to produce to a specific outcome
- Examples from CSE 12X: binary search, merge sort, recursive backtracking

Abstract Data Types (ADT)

Abstract Data Type

- A type of organization of data that we define through its behavior and operations
  - Defines the input and outputs, not the implementations

Invented in 1974 by Barbara Liskov

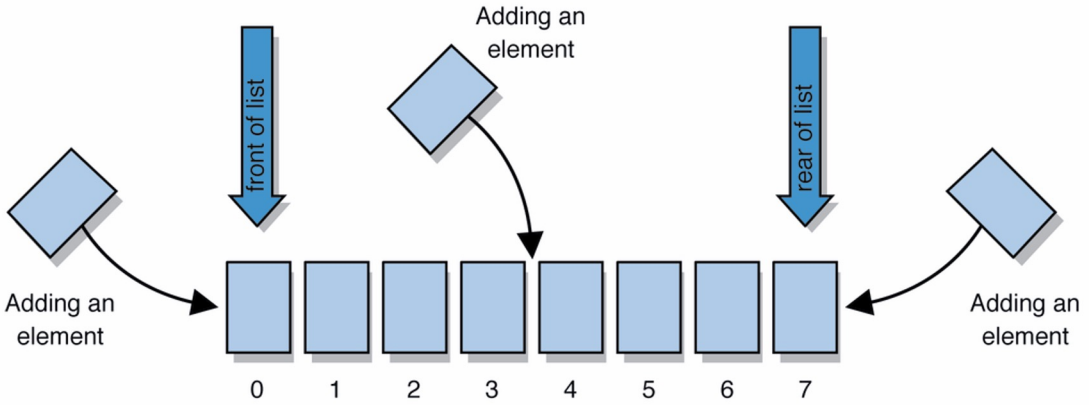
> *What we desire from an abstraction is a mechanism which permits the expression of relevant details and the suppression of irrelevant details. In the case of programming, the use which may be made of an abstraction is relevant; the way in which the abstraction is implemented is irrelevant. — Barbara Liskov*

Source Article

Abstract Data Types (ADT): List Example

Review: List – a collection storing an ordered sequence of elements

- each element is accessible by a 0-based index
- a list has a size (number of elements that have been added)
- elements can be added to the front, back, or elsewhere
- the ADT of a list can be implemented many ways through different Data Structures
  - in Java, a list can be represented as an ArrayList object



Review: Interfaces

interface: a construct in Java that defines a set of methods that a class promises to implement

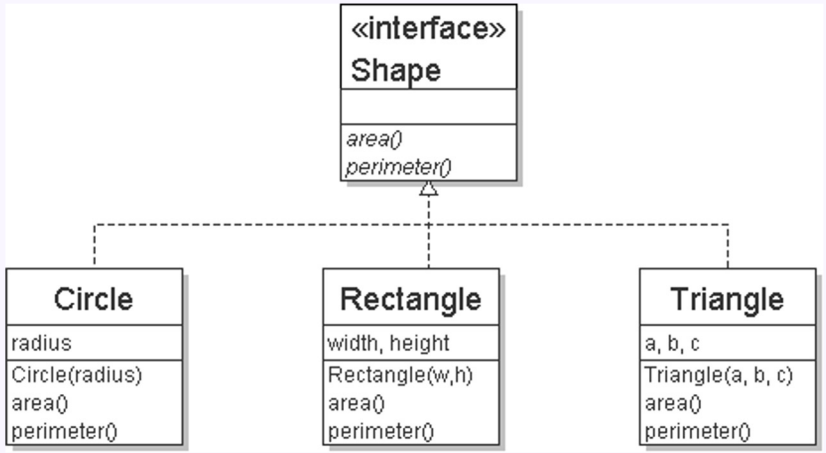
```
public interface name {
    public type name(type name, …, type name );
    public type name(type name, …, type name );
    …
}
```

- Interfaces give you an is-a relationship *without* code sharing.
  - A `Rectangle` object can be treated as a `Shape` but inherits no code.
- Analogous to non-programming idea of roles/certifications:
  - "I'm 'certified' as a Shape, because I implement the Shape interface. This assures you I know **how** to compute my area and perimeter."
  - "I'm 'certified' as a CPA accountant. This assures you I know **how** to do taxes, audits, and consulting."

Example

```
// Describes features common to all
// shapes.

public interface Shape {
      public double area();
      public double perimeter();
}
```



Review: Interfaces: List Example

interface: a construct in Java that defines a set of methods that a class promises to implement

In terms of ADTs, interfaces help us make sure that our implementations of that ADT are doing what they need to

For example, we could define an interface for the ADT `List<E>` and any class that implements it must have implementations for all of the defined methods

```
// Describes features common to all lists.

public interface List<E> {
    public E get(int index);
    public void set(E element, int index);
    public void append(E element);
    public E remove(int index);
    …

    // many more methods
}
```

note: this is not how the `List<E>` interface actually looks, as in reality it extends another interface

Review: Java Collections

Java provides some implementations of ADTs for you!

ADTs	Data Structures
Lists	`List<Integer> a = new ArrayList<Integer>();`
Stacks	`Stack<Character> c = new Stack<Character>();`
Queues	`Queue<String> b = new LinkedList<String>();`
Maps	`Map<String, String> d = new TreeMap<String, String>();`

But some data structures you made from scratch… why?

Linked Lists – LinkedIntList was a collection of ListNode
Binary Search Trees – SearchTree was a collection of SearchTreeNodes

Full Definitions

Abstract Data Type (ADT)

- *A definition for expected operations and behavior*
- A mathematical description of a collection with a set of supported operations and how they should behave when called upon
- Describes what a collection does, not how it does it
- Can be expressed as an interface
- Examples: List, Map, Set

Data Structure

- *A way of organizing and storing related data points*
- An object that implements the functionality of a specified ADT
- Describes exactly how the collection will perform the required operations
- Examples: LinkedIntList, ArrayIntList

ADTs and Data Structures: (Loose) Analogy

Abstract Data Type (ADT)

Mode of Transportation
Must be able to move
Must be able to be steered

Data Structures

Car	**Airplane**	**Bike**
Tires	Engines/wings	Wheels
Steering wheel	Control column	Handlebars

ADTs and Data Structures: List Example

List – Abstract Data Type (ADT)

```
// Describes features common to all lists.

public interface List<E> {
    public E get(int index);
    public void set(E element, int index);
    public void append(E element);
    public E remove(int index);
    …
}
```

ArrayIntList – Data Structure

```
public class ArrayIntList extends List<E>{
        private int[] list;
        private int size;

    public ArrayIntList(){
        //initialize fields
    }

    public int get(int index){
        return list[index];
    }

    public void set(E element, int index){
        list[index] = element;
    }
    …
}
```

ADTs we'll discuss this quarter

- List: an ordered sequence of elements
- Set: an unordered collection of elements
- Map: a collection of "keys" and associated "values"
- Stack: a sequence of elements that can only go in or out from one end
- Queue: a sequence of elements that go in one end and exit the other
- Priority Queue: a sequence of elements that is ordered by "priority"
- Graph: a collection of points/vertices and edges between points
- Disjoint Set: a collection of sets of elements with no overlap