Abstract Data Types

Abstract Data Types

- An **abstract data type** is a mathematical set of data, along with operations defined on that kind of data
- Examples:
  - **int**: it is the set of integers (up to a certain magnitude), with operations +, -, /, *, %
  - **double**: it's the set of decimal numbers (up to a certain magnitude), with operations +, -, /, *

Data Structures

- A data structure is a user-defined abstract data type
- Examples:
  - **Complex numbers**: with operations +, -, /, *, *magnitude*, *angle*, etc.
  - **Stack**: with operations *push*, *pop*, *peek*, *isempty*
  - **Queue**: *enqueue, dequeue, isempty …*
  - **Binary Search Tree**: *insert*, *delete*, *search.*
  - **Heap**: *insert*, *min*, *delete-min*.

Data Structure Design

- Specification
  - A set of data
  - Specifications for a number of operations to be performed on the data
- Design
  - A lay-out organization of the data
  - Algorithms for the operations
- Goals of Design: **fast** operations

Implementation of a Data Structure

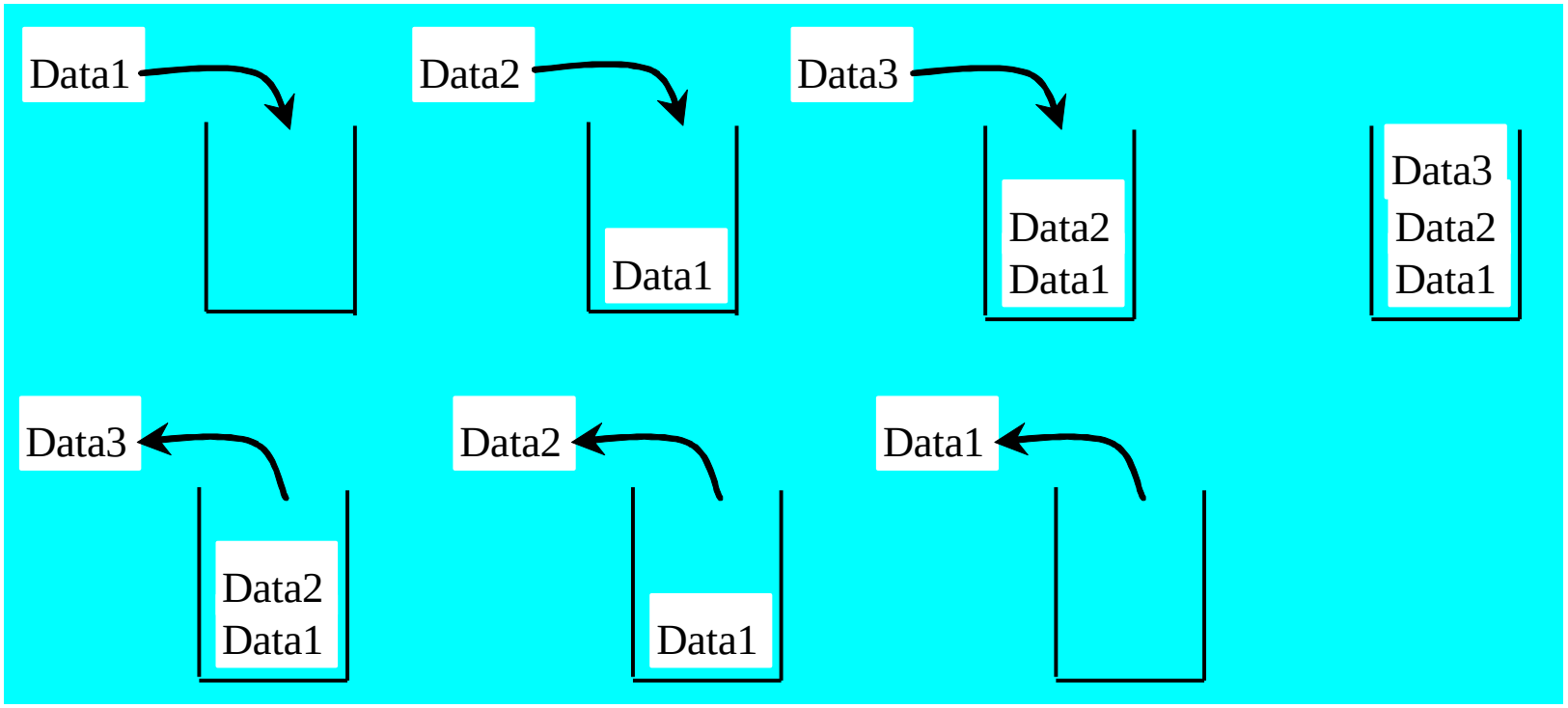
- Representation of the data using built-in data types of the programming language (such as int, double, char, strings, arrays, structs, classes, pointers, etc.)
- Language implementation (code) of the algorithms for the operations

- In OOP languages both the data representation and the operations are aggregated together into what is called **objects**
- The data type of such objects are called **classes**.
- Classes are blue prints, objects are instances.

Stack, Queue and List

Stacks

A stack can be viewed as a special type of list, where the elements are accessed, inserted, and deleted only from the end, called the top, of the stack.

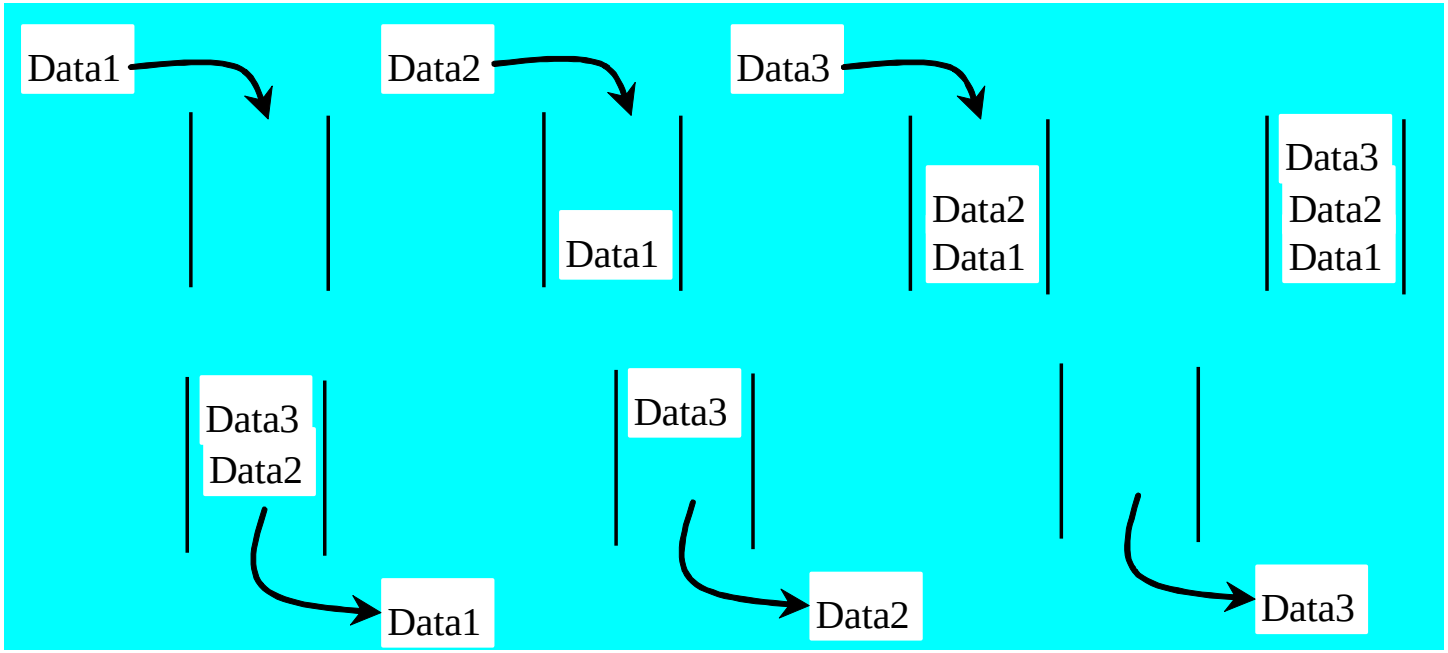


Stack

- A stack is maintained Last-In-First-Out
  (not unlike a stack of plates in a cafeteria)
- Standard operations
  - `isEmpty()` : returns `true` or `false`
  - `top()` : returns copy of value at top of stack (without removing it)
  - `push(v)` : adds a value v at the top of the stack
  - `pop()` : removes and returns value at top

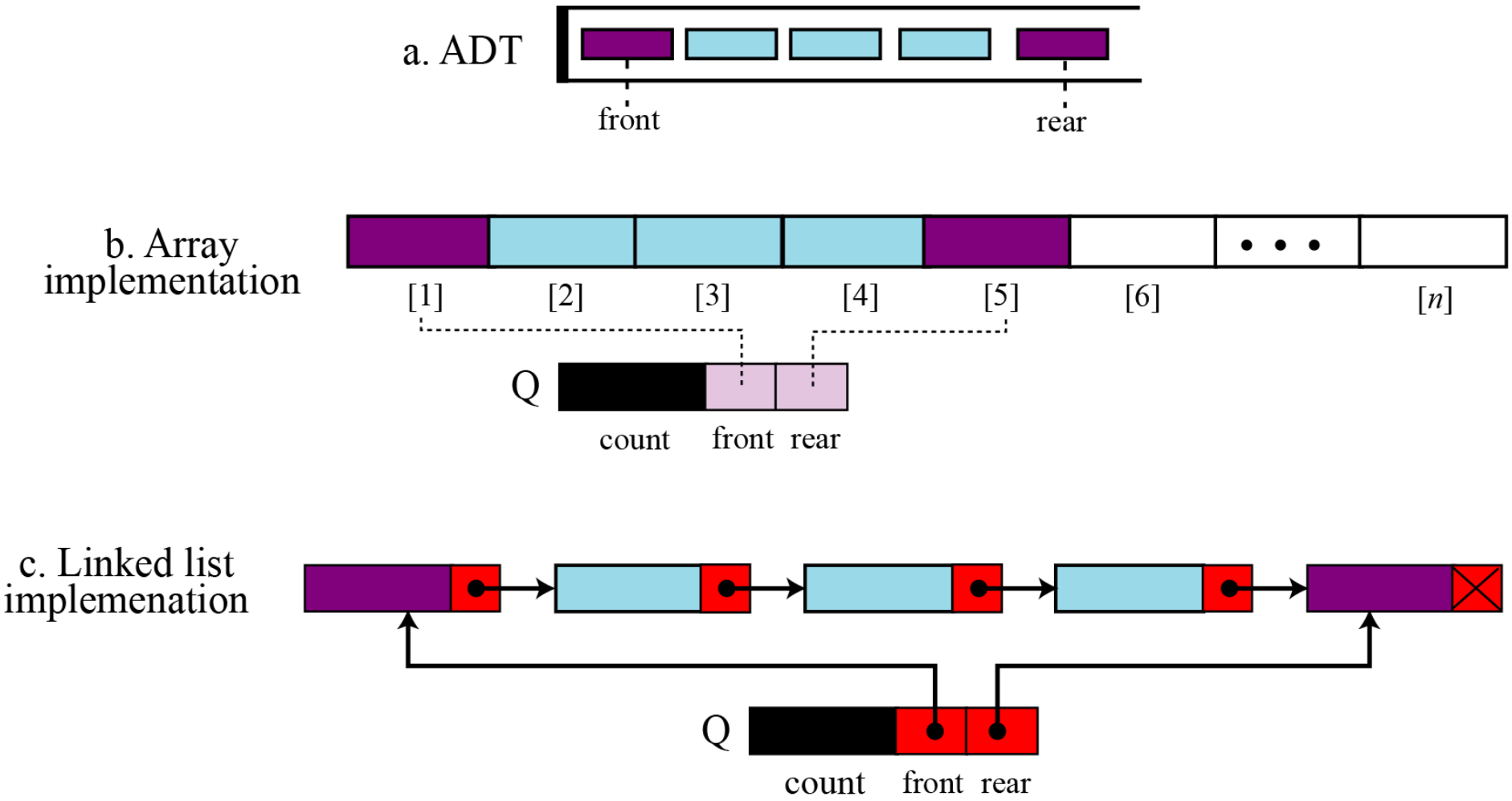
Queues

A queue represents a waiting list. A queue can be viewed as a special type of list, where the elements are inserted into the end (tail) of the queue, and are accessed and deleted from the beginning (head) of the queue.



Queues

- Queue Manipulation Operations
  - `isEmpty()` : returns `true` or `false`
  - `first()` : returns copy of value at front
  - `add(v)` : adds a new value at rear of queue Enqueue
  - `remove()` : removes, returns value at front Dequeue



Queue implementation

Implementing Stacks and Queues

- Using an Arraylist to implement Stack
- Use a Linked list to implement Queue

Since the insertion and deletion operations on a stack are made only at the end of the stack, using an array list to implement a stack is more efficient than a linked list.

Since deletions are made at the beginning of the list, it is more efficient to implement a queue using a linked list than an array list.

Design of the Stack and Queue Classes

There are two ways to design the stack and queue classes:

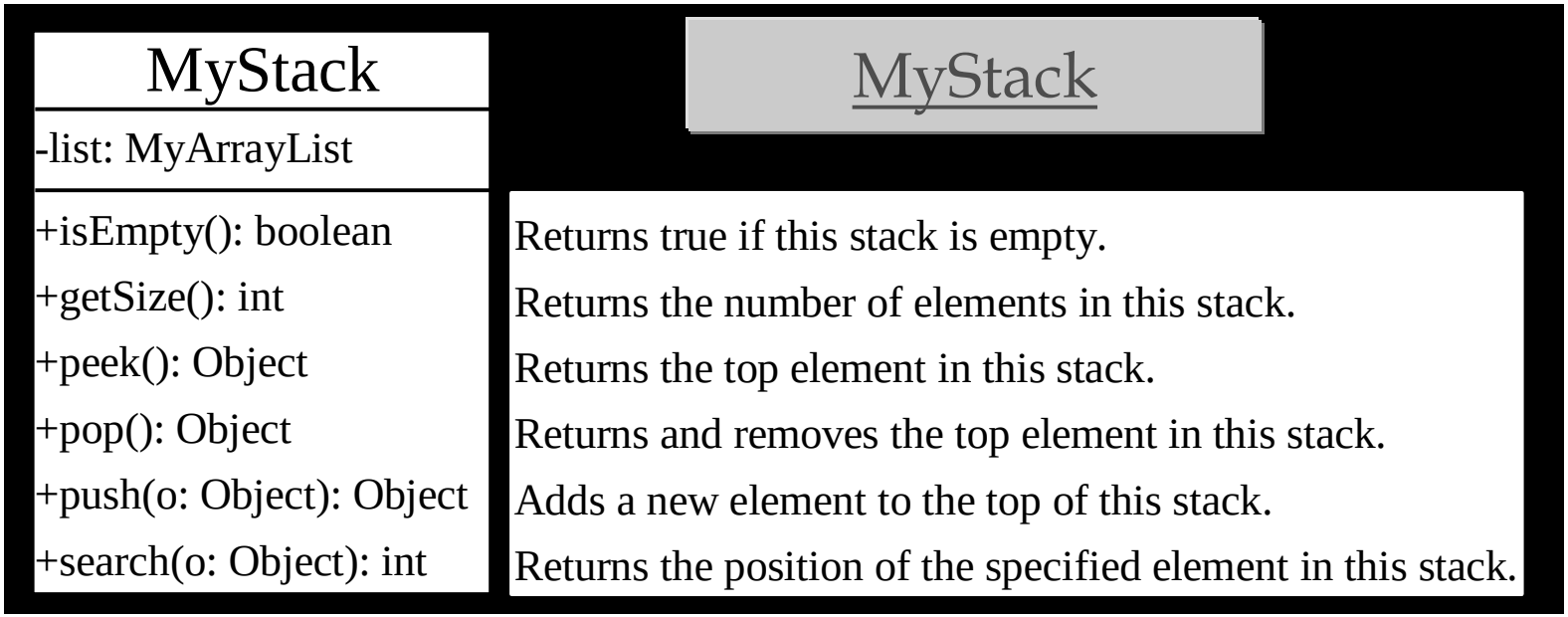
- Using inheritance: You can declare the stack class by extending the array list class, and the queue class by extending the linked list class.

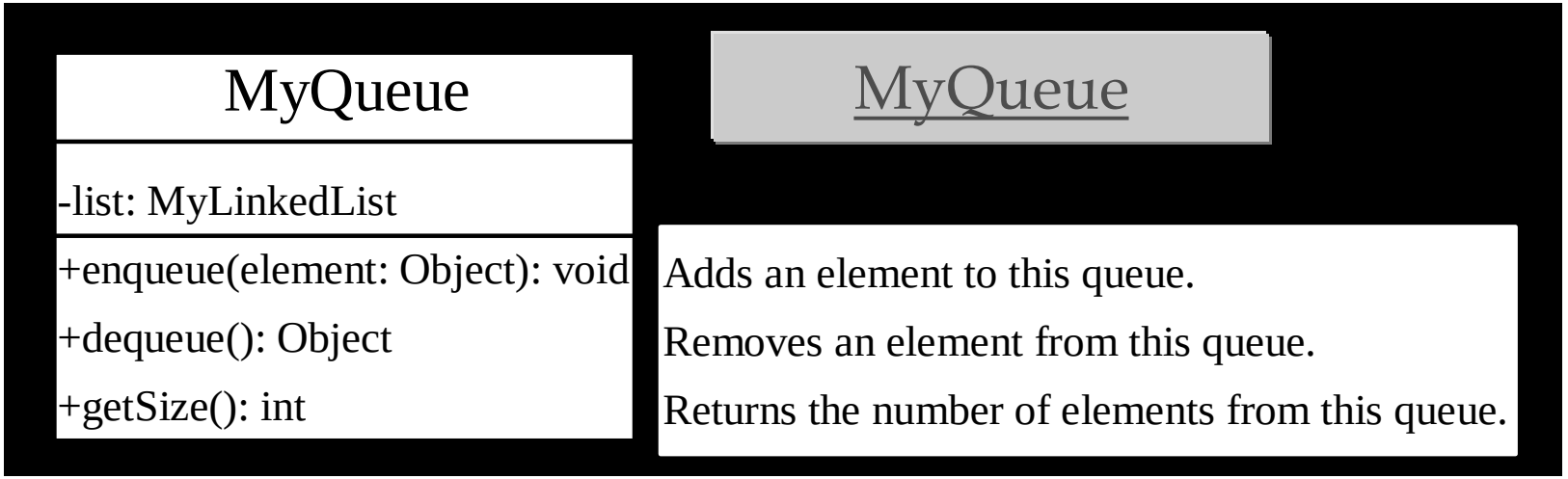
  - Using composition: You can declare an array list as a data field in the stack class, and a linked list as a data field in the queue class.

Composition is Better

Both designs are fine, but using composition is better because it enables you to declare a complete new stack class and queue class without inheriting the unnecessary and inappropriate methods from the array list and linked list.

MyStack and MyQueue

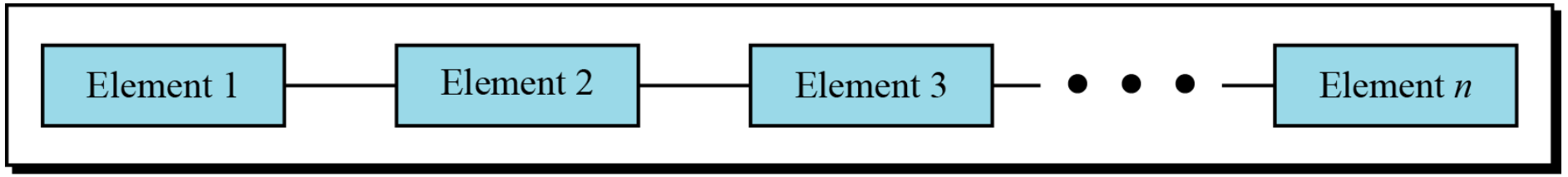




GENERAL LINEAR LISTS

•Stacks and queues defined in the two previous sections are restricted linear lists.

•A general linear list is a list in which operations, such as insertion and deletion, can be done anywhere in the list—at the beginning, in the middle or at the end. Figure shows a general linear list.



General linear list

Operations on general linear lists

Six common operations: *list*, *insert*, *delete*, *retrieve*, *traverse* and *empty*.

The *list* operation

The list operation creates an empty list. The following shows the format:

list (listName)

List features

- **ORDERING**: maintains order elements were added (new elements are added to the end by default)
- **DUPLICATES**: yes (allowed)
- **OPERATIONS**: add element to end of list, insert element at given index, clear all elements, search for element, get element at given index, remove element at given index, get size
  - some of these operations are inefficient! (seen later)

- list manages its own size; user of the list does not need to worry about overfilling it

Java's `List` interface

- Java also has an interface `java.util.List` to represent a list of objects:
  (a partial list)

```
public void add(int index, Object o)
```

Inserts the specified element at the specified position in this list.

```
public Object get(int index)
```

Returns the element at the specified position in this list.

```
public int indexOf(Object o)
```

Returns the index in this list of the first occurrence of the specified element, or `-1` if the list does not contain it.

`List` interface, cont'd.

`public int lastIndexOf(Object o)`

Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain it.

`public Object remove(int index)`

Removes the object at the specified position in this list.

`public Object set(int index, Object o)`

Replaces the element at the specified position in this list with the specified element.

- Notice that the methods added to `Collection` by `List` all deal with indexes; a list has indexes while a general collection may not

Some list questions

- all of the list operations on the previous slide could be performed using an array instead!

- open question: What are some reasons why we might want to use a list class, rather than an array, to store our data?

- thought question: How might a List be implemented, under the hood?

- why do all the List methods use type `Object`?