COMP 202 – Week 14

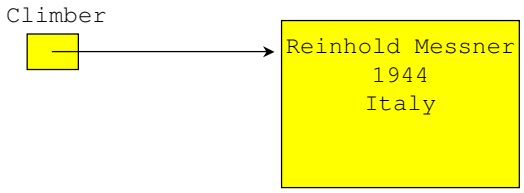
- Next, we explore some advanced techniques for organizing and managing information
- This week we focus on:
  - dynamic structures
  - Abstract Data Types (ADTs)
  - linked lists
  - trees
  - queues

Static vs. Dynamic Structures

- A *static* data structure has a fixed size
- This meaning is different than those associated with the `static` modifier
- Arrays are static; once you define the number of elements it can hold, it doesn't change
- A *dynamic* data structure grows and shrinks as required by the information it contains

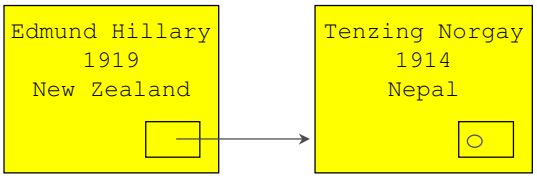
Object References

- Recall that an *object reference* is a variable that stores the address of an object
- A reference can also be called a *pointer*
- They are often depicted graphically:



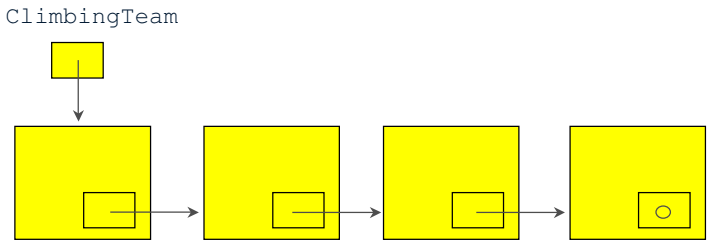
References as Links

- Object references can be used to create *links* between objects
- Suppose a `Climber` class contained a reference to another `Climber` object



References as Links

- **References can be used to create a variety of linked structures, such as a *linked list*:**



Abstract Data Types

- **An *abstract data type* (ADT) is an organized collection of information and a set of operations used to manage that information**
- **The set of operations define the *interface* to the ADT**
- **As long as the ADT accurately fulfills the promises of the interface, it doesn't really matter how the ADT is implemented**
- **Objects are a perfect programming mechanism to create ADTs because their internal details are *encapsulated***

Abstraction

- Our data structures should be abstractions
- That is, they should hide details as appropriate
- We want to separate the interface of the structure from its underlying implementation
- This helps manage complexity and makes the structures more useful

Intermediate Nodes

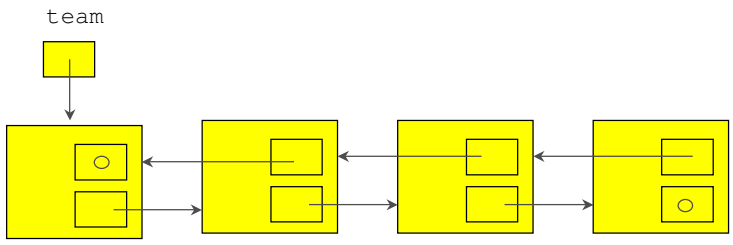
- The objects being stored should not have to deal with the details of the data structure in which they may be stored
- For example, the `Climber` class should not store a link to the next `Climber` object in the list
- Instead, we can use a separate node class that holds a reference to the stored object and a link to the next node in the list
- Therefore the internal representation actually becomes a linked list of nodes

A collection of climbers

- Let's explore an example of a collection of `Climber` objects
- The collection is managed by the `ClimbingTeam` class, which has an private inner class called `ClimberNode`
- Because the `ClimberNode` is private to `ClimbingTeam`, the `ClimbingTeam` methods can directly access `ClimberNode` data without violating encapsulation
- See `ClimbingTeamIsEmptyException.java`
- See `Climber.java`
- See `ClimbingTeam.java`
- See `K2Ascent.java`

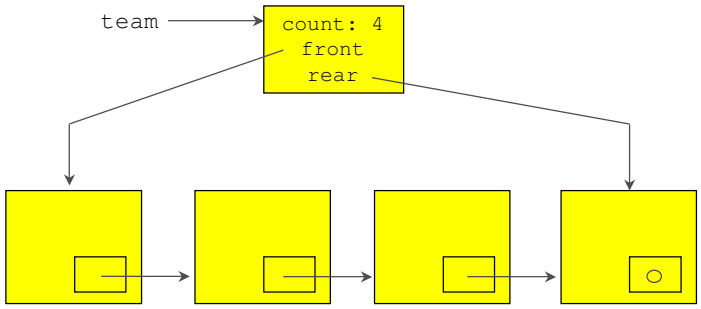
Other Dynamic List Implementations

- It may be convenient to implement as list as a *doubly linked list*, with next and previous references:



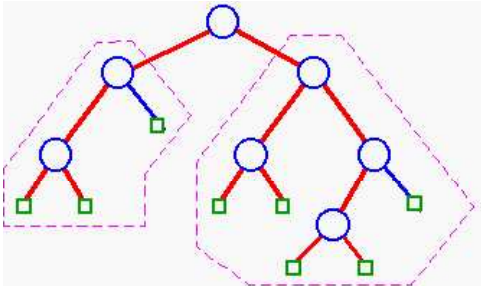
Other Dynamic List Implementations

- It may also be convenient to use a separate header node, with references to both the front and rear of the list

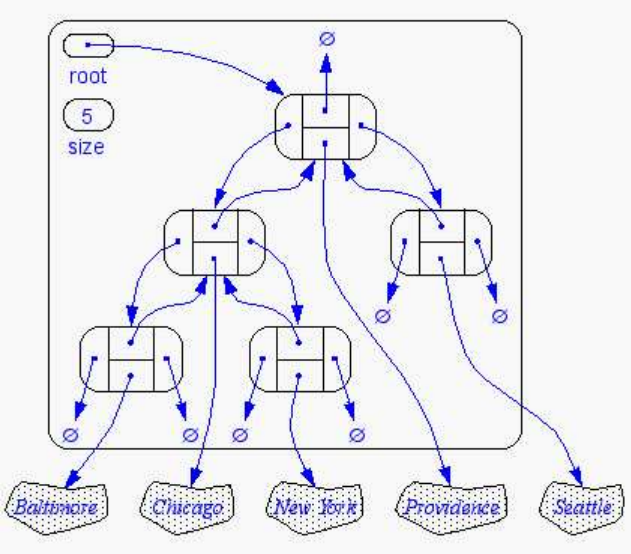


Trees

- A *tree* is a data structure that represents a hierarchy, through internal and external nodes
- Ex: table of contents for a book, OS file system, inheritance relationship between Java classes, organizational structure of a corporation, etc.
- A *binary tree* is a tree where each internal node has exactly 2 child nodes. A Binary tree is either (recursive definition):
  - An external node (a *leaf*)
  - A internal node and two binary trees (left subtree and right subtree)

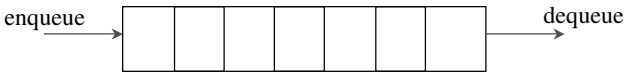


Linked Tree Implementation



Queues

- A *queue* is similar to a list but adds items only to the end of the list and removes them from the front
- It is called a FIFO data structure: First-In, First-Out
- Analogy: a line of people at a bank teller's window



Queues

- We can define the operations on a queue as follows:
  - enqueue - add an item to the rear of the queue
  - dequeue - remove an item from the front of the queue
  - empty - returns true if the queue is empty
- As with our linked list example, by storing generic `Object` references, any object can be stored in the queue
- Queues are often helpful in simulations and any processing in which items get "backed up"

Two Queue ADT implementations

- See `Queue.java`
- See `ArrayQueue.java`
- See `LinkedQueue.java`
- See `Farewell.java`