Chapter 12: Data Structures

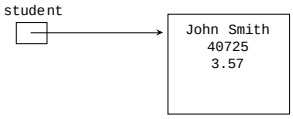
- Let's explore some advanced techniques for organizing and managing information
- Chapter 12 focuses on:
  - dynamic structures
  - Abstract Data Types (ADTs)
  - linked lists
  - queues
  - stacks

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

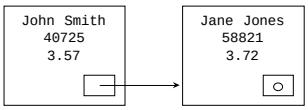
Object References

- 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:



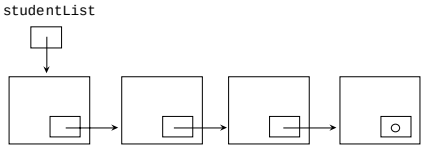
References as Links

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



References as Links

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



Abstraction

- Our data structures should be abstractions
- That is, they should hide details as appropriate
- This helps manage complexity
- Our original Library solution is not abstract
- One problem is that we use a public variable to set the links

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*

Coupling and Cohesion

- A well-defined ADT attempts to minimize coupling while maximizing cohesion
- Coupling is the strength of the relationship between two components
- Cohesion is the strength of the relationships among the parts of one component
- We want to formally specify a simple relationship between the ADT and the outer world, and put only those things that relate to the ADT management inside

Library

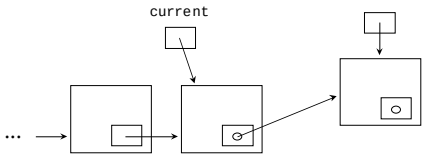
- A version of the Library solution would create a `BookList` object that provides services such as:
  - add a book to the list
  - print the book list
- The `BookList` object in turn interacts with individual `Book` objects
- Each `Book` object governs its own references privately
- See `Library.java`

Library Revisited

- The `Book` class does not contain the reference to the next object in the list
- Ideally, we would like to separate the data structure management from the information it holds
- The objects we want to store in the list should not have to be involved with the list references

Adding a Node

- To add a new node to the end of a linked list, traverse the list looking for the last node, then add the new object
- The last node is the one with a null `next` reference

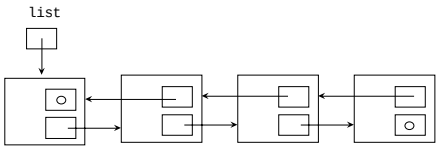


Other List Operations

- You may also want to perform such operations as:
  - add a node to the front of the list
  - add a node somewhere in the middle of the list
  - delete a node from the list
- Each operation can be defined separately as its own method
- How an operation is implemented depends on the underlying representation of the data structure

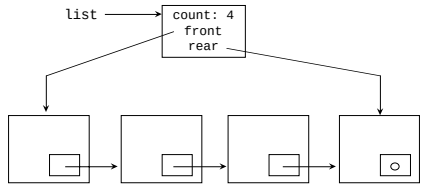
Other Dynamic List Implementations

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



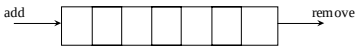
Other Dynamic List Implementations

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



Queues

- A *queue* ADT 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

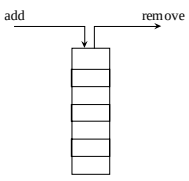
- We can define the operations on a queue as follows:
  - enqueue - add an item to the rear of the queue
  - dequeue - remove an item from the front of the queue
  - empty - returns true if the queue is empty
- By operating on the `Object` class, any object can be stored in the queue

Stacks

- A *stack* ADT is also linear, like a list or queue
- Items are added and removed from only one end of a stack
- It is therefore LIFO: Last-In, First-Out
- Analogy: a stack of plates

Stacks

- Stacks are often drawn vertically:



Stacks

- Some stack operations:
  - push - add an item to the top of the stack
  - pop - remove an item from the top of the stack
  - peek - retrieves the top item without removing it
  - empty - returns true if the stack is empty
- The `java.util` package contains a `Stack` class, which is implemented using a `Vector`
- See `Decode.java`