Abstractions, Collections & Data Structures

CMSC 132

Department of Computer Science

University of Maryland, College Park

Overview

- Abstractions and Collections
- Collections and Data Structures
- Java Collections Framework
- Collection Interface
- Collection Hierarchy
- Linear Data Structures
- Operations on Lists
- Accessing Elements
- Restricted Abstractions
- Queue
- Stack
- General List Operations
- List Implementations

Abstractions and Collections

- Programs represent and manipulate abstractions, or chunks of information
  - A Sudoku board
    - the moves legal for a particular square
  - A media player
  - A deck of cards
- The most universal of these is a collection
  - Lots of different kinds of collections
    - list, set, ordered set, bag, map
  - For each kind of collection, there are several operations you can perform on them

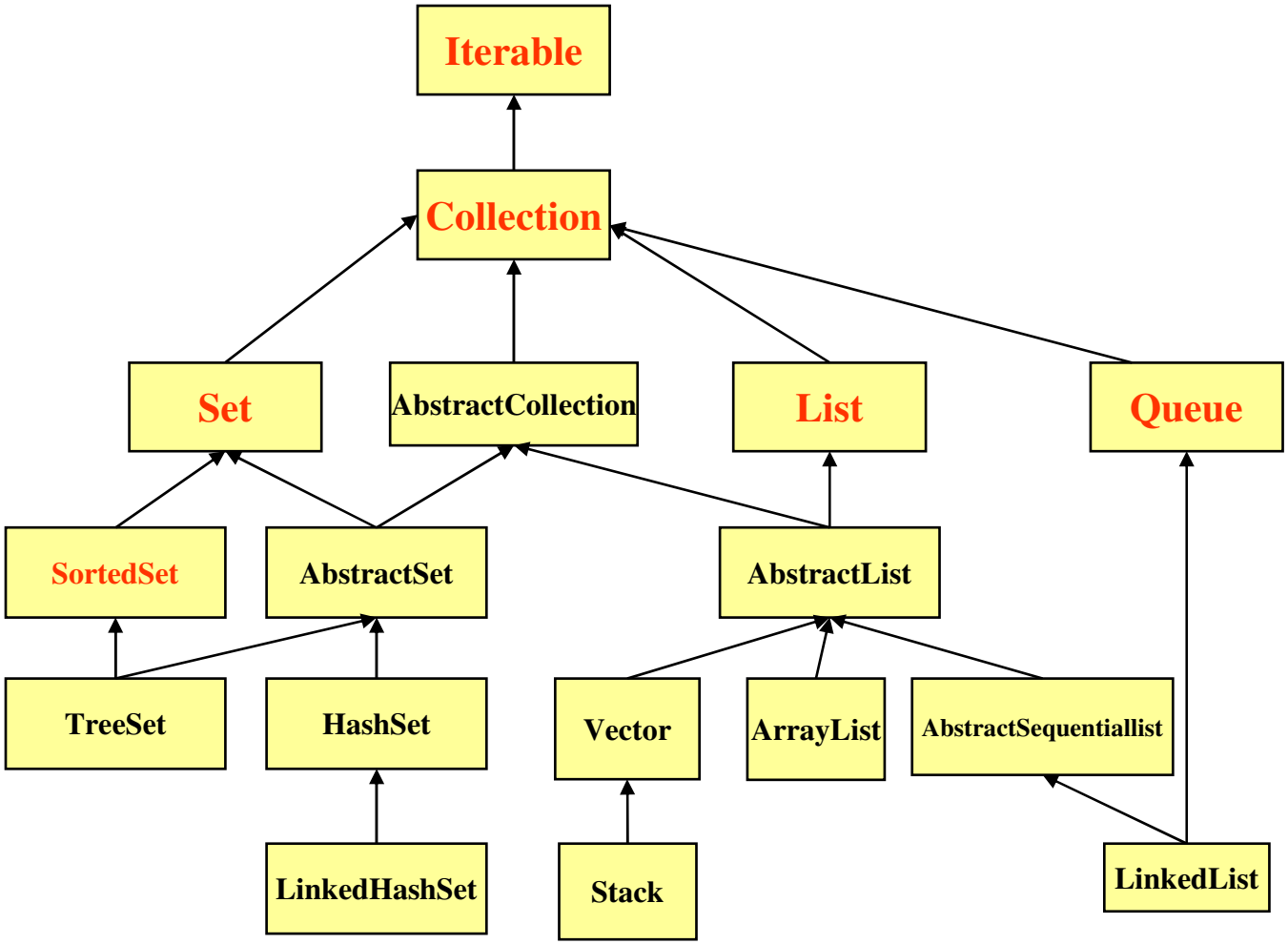
Collections and Data Structures

- A collection can typically be implemented using several different data structures
  - A data structure is a way of implementing an abstraction and operations on it
- Different implementations or data structures for an abstraction will provide different efficiencies for different operations
  - We will see more of this in a moment

Java Collections Framework

- Java provides several interfaces and classes that allow us to manipulate/organize data
- These group of interfaces and classes is known as the Java Collections Framework (JCF)
- JCF supports three types of collections
  - set – defined in the Set interface
  - list – defined in the inter List interface
  - map – (associated array) defined in the Map interface
- All JCF interfaces and classes are part of the java.util package
- First we will look at sets and lists
- All concrete classes in the JCF can be cloned and serialized.

Collection Hierarchy



- Interfaces – represented by red font
- Classes – represented by black font

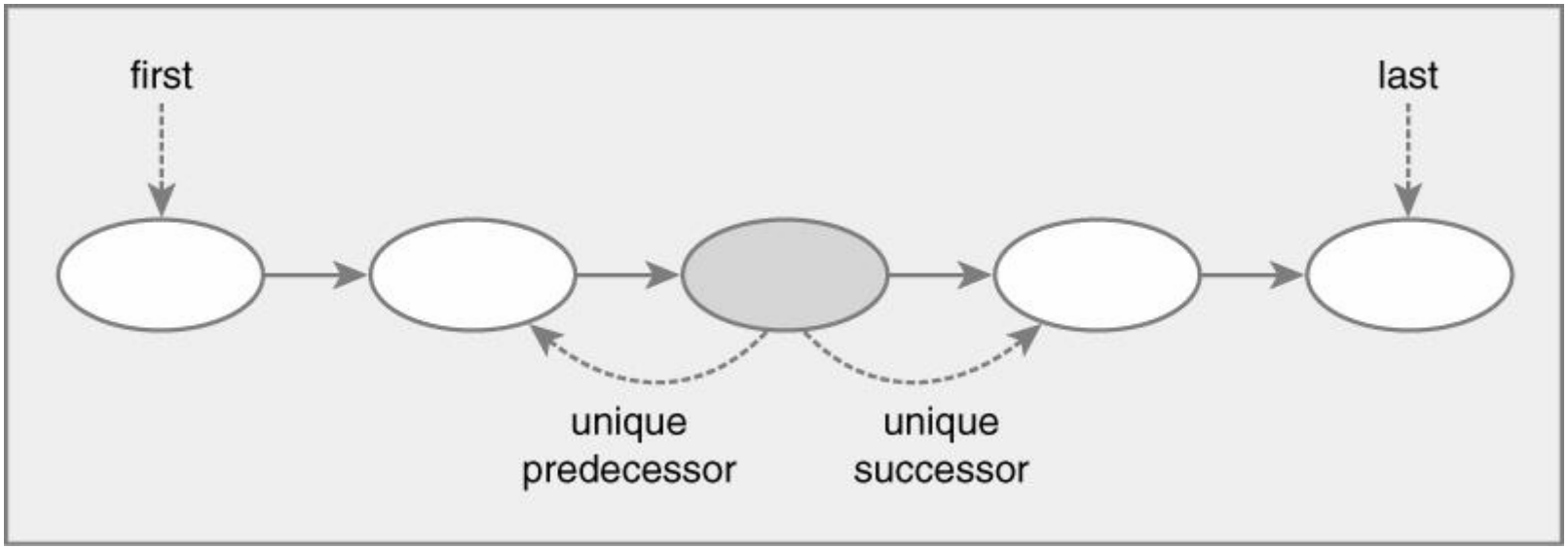
Collection Interface

- The base interface of most collections
- Operations supported:
  - add elements
  - remove elements
  - determine size of collection (# of elements)
  - iterate through elements
- Collection interface can be confused with the Collections (extra s) class
- Collections class – class that consists of static methods that operate on collections
- Java API Entry

  http://java.sun.com/j2se/1.5.0/docs/api/java/util/Collection.html

Linear Data Structures

- **Total order over elements**
  - **Each element has unique predecessor**
  - **Each element has unique successor**
  - **For any two distinct elements x and y, either x comes before y or y comes before x**



Linear Data Structures

- **Core operations**
  - Find first element (head)
  - Find next element (successor)
  - Find last element (tail)
- **Terminology**
  - Head $\Rightarrow$ no predecessor
  - Tail $\Rightarrow$ no successor
- **Duplicates allowed**
  - The same value can appear at different places in the list

Operations on Lists

- Add an element
  - Where?
    - at beginning/front
    - at rear/end
    - after a particular element
- Remove an element
  - remove first element
  - remove last element
  - remove the String value “Happy”
    - what happens if “Happy” occurs more than once

Accessing Elements

- How do you name an element you want to manipulate?
  - first/head
  - last/tail
  - by position (e.g, the 5th element)
    - also a bad movie ☺
  - by walking/iterating through the next, and talking about relative position
    - the next element
    - the previous element

Restricted Abstractions

- Restricting the operations an abstraction supports can be a good thing
  - efficiently supporting only a few operations efficiently is easier
  - if a limited abstraction suits your needs, easily to reason about the limited abstraction than a more general one
- Restricted list abstractions:
  - queue (aka FIFO queue)
  - stack (aka LIFO queue)
  - dequeue (aka double ended queue)

Queue

- Can add items to the end of the queue
- Remove them from the front of the queue
- First-in, first-out order (FIFO)
- Represented as a list
  - total order over elements
- But no access to middle of the list
- Example use:
  - songs to be played
  - jobs to be printed
  - customers to be served
- See QueueDemo.java in code distribution

Queue

- Java defines a Queue interface
- Java API Entry

  http://java.sun.com/j2se/1.5.0/docs/api/java/util/Queue.html
- One of the classes implementing the interface is the LinkedList

Stack

- Can add an item to the top of the stack
- Can remove the item on top of the stack
- No access to elements other than the top
- Just a list where you can add/remove elements only at one end
  - whether the top of the stack is the front or beginning of the list doesn't really matter
- Examples:
  - keep track of pending calculations in a calculator
  - Parsing
- Java API entry

  http://java.sun.com/j2se/1.5.0/docs/api/java/util/Stack.html

Double ended queue

- List/queue where you can add or remove at either end

General List Operations

- get/set/insert/remove value at a particular index
- get/set/insert/remove value at head/tail
- iterate through list, and get/set/insert/remove values as you go
  - use a java.util.ListIterator

List Implementations

- **Two basic implementation techniques for any kind of list**
  - **Store elements in an array**
  - **Store each element in a separate object, and link them together (a linked list representation)**

List Implementation: Array

- **Advantages:**
  - Space efficient (just space to hold reference to each element)
  - Can efficiently access elements at any position
- **Disadvantages**
  - Has to grow/shrink in spurts
  - Can't efficiently insert/remove elements in the middle
  - inserting/removing elements at both ends is tricky

List Implementation: Linked

- **Advantages**
  - Can efficiently insert/remove elements anywhere
- **Disadvantages**
  - Can't efficiently access elements at arbitrary positions
  - Space overhead (one or two additional pointers per element)

Examples

- See ListExample1.java and ListExample2.java in code distribution