

# Java Collections Framework (JCF): Lists, Stacks, Queues, Priority Queues, Sets, and Maps

# Administrivia

- Lab 5 graded
- Lab 6 grades coming soon
- Project 2 due on Wednesday
  - Testing
  - Exceptions
  - Code review slots open
- Lab 8 released soon
- Project 3 coming this week
- Exam 1

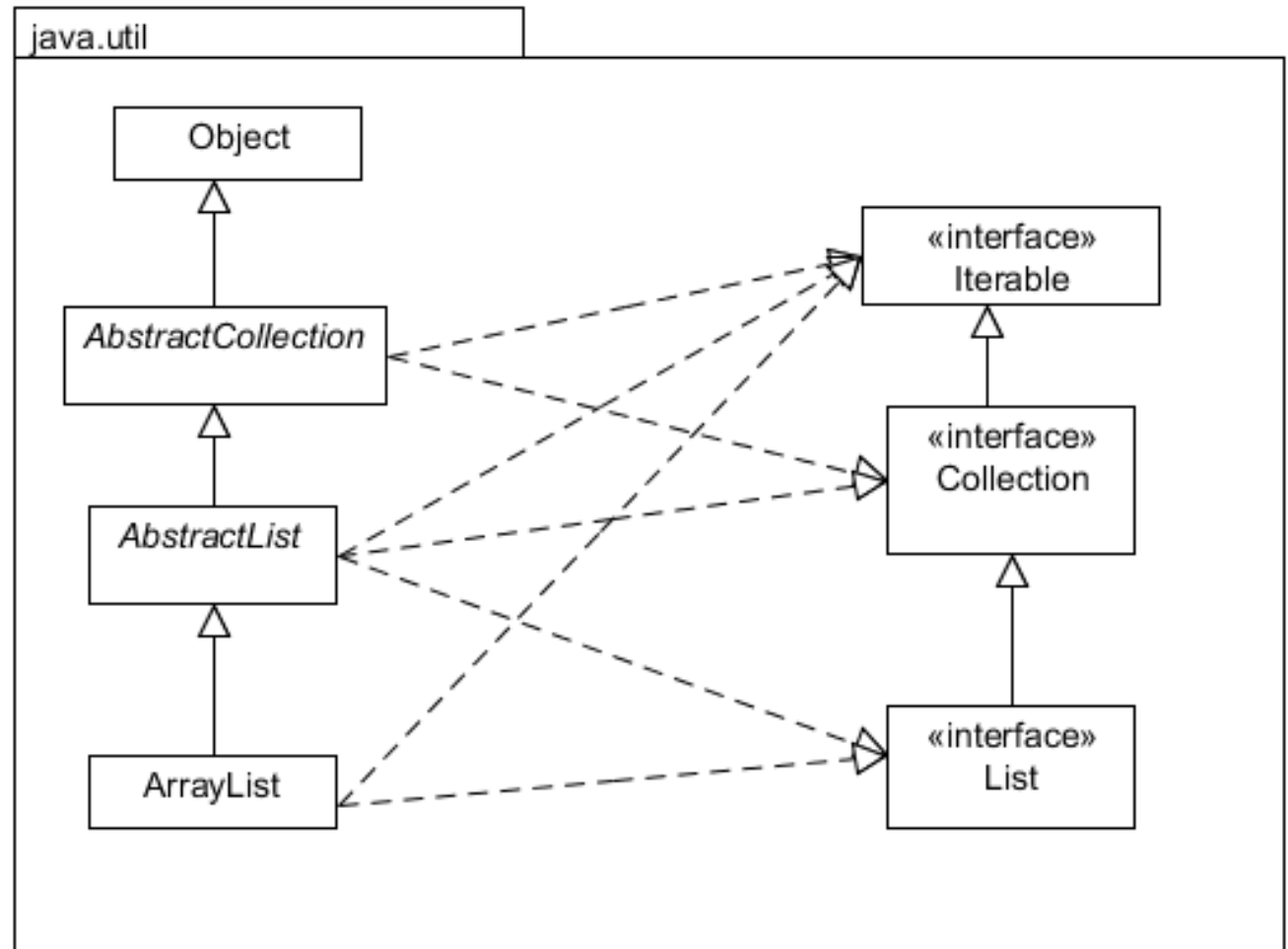
# Data Structures vs Abstract Data Types

- Data Structure:
  - A specific way of organizing data and operations to access/use the data
  - Structure of the data tied directly to the implementation
- Abstract data type: An **implementation independent** group of data and a set of operations on this data

# Data Structures We Know

- Both arrays and ArrayList are data structures
  - Implementation dependent
- What are the similarities between ArrayList and arrays?
  - Organization of data?
  - Operations?
- What are the differences between these?
  - Organization of data?
  - Operations?

# Class Hierarchy for ArrayList



- Which are abstract data types?
- Which are data structures?

# Collection Interface

What is a collection?

# Collection Interface

What is a collection?

- “Bag” (or “pile”) of objects

# Collection Interface

What is a collection?

- “Bag” (or “pile”) of objects (no ordering)

What does the Collection interface provide?

- add()
- remove()
- contains()
- Iterator: a means of iterating over all objects in the collection



# List Interface

- What is a list?
  - Ordered Collection
  - Duplicates

# List Interface

- What is a list?
  - Ordered Collection
  - Duplicates
- Java API: List interface
  - What else is provided over a Collection?

# Linked Lists

- Example of a linked list using people ...
  - Singly linked list versus doubly linked list
  - How do we search for items?
  - How efficient is it to add items?
- Java API: examine LinkedList API

# LinkedList

- LinkedList: concrete class
  - What methods are in LinkedList and not ArrayList?
- Critical data structure difference: LinkedList vs. ArrayList
  - Incremental allocation
  - LinkedList makes adding to the head and tail of the list cheaper

# Choosing Lists

A LinkedList is used instead of an ArrayList when:

- Size of structure changes radically over time
  - Once ArrayLists get big, they stay big
- Random access not needed
  - What does this do to binary search?
- Insertion and deletion at head and tail and more common than search

# Administrivia

- Project 2 should be complete
  - Code reviews due in 1 week
  - Sign up for a slot or come in for a “walk-in” review
- Lab 8 is available and due on Monday @7pm
- Project 3 will be out by the weekend (with discussion on Monday)

# Iterator<E>

Object that iterates over some collection

- `hasNext()`: is there another item in the collection that hasn't been "touched"
- `next()`: return the next item

# LinkedList Example

Linked List of Person...



# Iterator<E>

- In general, you should not be modifying the list as you are iterating over it!
- Can lead to very strange behavior
- Exception: Iterator<E> provides a remove() method

# Collection vs Collections

- **Collection interface**: The root of the JCF hierarchy
  - Represent a group of objects
  - Operations include: add/remove/iterate
- **Collections class**: provides many static methods, including: shuffle, max, min, reverseOrder, sort, frequency, ...

Examine API for Collections and Collection...

# Person example II

- Sort the Persons
  - Name, then ID
  - Reverse order by ID
- Use an Iterator to loop through the scores
  - Explicit Iterator
  - Implicit Iterator (for each loop)

# Abstract Data Type: Queue

- Example of queue with people to buy tickets
- Key: First in, First out (FIFO)
- See: Java API: Queue Interface

# Example

Store people about to compete in an event in Queue

# Priority Queue

- Standard Queue: order by insertion order
- Priority Queue: order by some ordering
  - Natural order or defined by a Comparator
- Java API: Examine PriorityQueue API
- Example: office hours
  - What happens if President Boren shows up?

# Stack

- Last in First out (LIFO)
- Add (push) and remove (pop) items to/from the top of the stack
- Data structure for Stack is extensible array

# Stack

- Example: grading exams
- Example: System stack
  - main() method is at the bottom
  - Most recent method call is at top
  - Call a new method: pushes the method onto the stack
  - Return from a method: pops the top method off of the stack



# Uses for Collections Class

- Shuffle exam questions on multiple choice tests
- Frequency to figure out how many coins of each type you have
- ReplaceAll to fix all of an item in an inventory (recalls etc)
- ReplaceAll to give everyone the same grade
- IndexOfSublist to return a sublist all students born in Boston assuming they are sorted by birth location
- nCopies to make lots of clones
- Empty to get rid of all your homework

# Backward Compatibility

- Vector and Stack
  - Part of Java from the start
  - Were retrofitted into the JCF
  - Synchronized— expensive, but needed for threading
- Vector
  - Data structure: extensible array
  - Added methods from List interface that weren't in class
  - Added generic
  - Uses Enumeration interface (old form of Iterator)

# Sets and Maps

# Set Interface

- Set is another abstract data type
  - Elements in a set are not ordered
  - No duplicate elements
- How could Set be implemented with an array data structure?
  - Why isn't this good enough?

# Set Interface

- Examine class hierarchy in API
  - Note similarities and differences to design for ArrayList hierarchy
- What operations are typical of sets in mathematics?
- What operations does Java Set support?
  - Which Java Set operations are similar to those in discrete math?

# Choosing Sets

- Sets are used when order isn't important
  - We're so used to using arrays, that we tend to think of order being important when it isn't
- Example: Bug tracking software
  - Store bug reports
  - Find bug reports
- Example: grocery list
- Remember sets are the theoretical basis of most of computer science—they are everywhere

# HashSet

HashSet is a data structure (also called a hash table) that implements the Set interface

# HashSet: Data Structure

## Approach:

- Create a hash code from the object
  - Summarizes the content of the object
  - May not be unique
  - Eclipse can generate automatically (\*\* demo)
- We've seen this method in the Object class
- Use the hash code as an address in a huge array (called a *hash table*)



# Example

- Create a set of students
  - What should our hash code be?
- Use set operations from Set Interface API
  - <http://docs.oracle.com/javase/tutorial/collections/interfaces/set.html>

# Example HashSet

- Suppose we're storing numerical data
- Address is  $\text{hashCode} \% \text{tableSize}$
- Let the table size be 100
  - Insert 23983, 10484, 3817692, 1968372, 938983
- Collision: move to the next free spot in the table
- Classic time/space tradeoff

# Hash Set

contains(Object o):

- Compute the hash value for the object
- Compute the address in the hash table
- Is anything at that address?
  - No: return false
  - Yes: is that object equal to o?

# Hash Set

contains(Object o):

- Compute the hash value for the object
- Compute the address in the hash table
- Is anything at that address?
  - No: return false
  - Yes: is that object equal to o?
    - Yes: return true

# Hash Set

contains(Object o):

- Compute the hash value for the object
- Compute the address in the hash table
- Is anything at that address?
  - No: return false
  - Yes: is that object equal to o?
    - Yes: return true
    - No: increment the hash table address



# Critical Hash Table Measurements

Load factor: # of used elements/table size

- Load factor needs to stay small for a table to work well
- When the load factor gets close to 1, clustering is a problem (many address increments)
- HashSet fixes this by reallocating the table when it gets too dense (expensive!)

# Critical Hash Table Measurements

Choosing the table size, the load factor, and the hash functions are critical parts to the success of hashing

- If these are done well, hashing is fabulous
- Lots of people don't use hashing because of fear of these factors
- **If managed correctly, hashing can be incredibly good**

# Example for Home

- Create a HashSet that stores 1000 randomly generated integers
- Search for each integer
- Measure time
  - `System.nanoTime()`
- Compare to ints stored in an ArrayList
  - How many lines of code have to be changed?
  - Compare what happens when table size properly created initially



# TreeSet

- The problem with HashSet is that elements aren't ordered in any useful way
  - How would you sort data in a hash table?
  - LinkedHashSet orders elements by time of entry, but this often isn't a useful order
- TreeSet uses a natural ordering (Comparable)
  - Can use alternate ordering (Comparator)

Note: we are breaking from the mathematical notion of set

# How Does TreeSet Work?

- Example brief introduction of Binary Search Tree
  - Tree balancing
- Will see full implementations in CS 2413 Data Structures

# Abstract Data Type: Map

Stores <key, value> pairs

- Key used to organize data (no duplicates)
  - Keys form a proper Set
- Value is the data itself (duplicates allowed)

# Map

Example: In computer gaming, all objects are stored in a Map <objectID, object>

- Objects are players, furniture, non-playable characters, etc.

# Map Interface

- Examine methods in API carefully
  - How would you get iteration?
- Examine class hierarchy in API
  - Lots of implementation options

# Example

- Implement a map that stores and retrieves names by an identification number
  - Use HashMap
- Examine differences in data ordering with HashMap, TreeMap, LinkedHashMap
  - HashMap used a lot in Java!

# Back to the Collections Class

- JCF stores its static methods in one shared class
- Examine which methods apply to which type
  - Why doesn't Set have sort/reverse?