CSSE 220 Day 17

Data-structure-palooza
Exam Review
Generics

Checkout *DataStructures* and *Generics2* from SVN

Questions

Data Structures

- Understanding the engineering trade-offs when storing data

Abstract Data Types

- Boil down data types (e.g., lists) to their essential operations
- Choosing a data structure for a project then becomes:
  - Identify the operations needed
  - Identify the abstract data type that most efficiently supports those operations
- Goal: that you understand several basic abstract data types and when to use them

Common ADTs

- Array List
- Linked List
- Stack
- Queue
- Set
- Map

Implementations for all of these are provided by the Java Collections Framework in the **`java.util`** package.

Array Lists and Linked Lists

Operations Provided	Array List Efficiency	Linked List Efficiency
Random access	O(1)	O(n)
Add/remove item	O(n)	O(1)

Stacks

- A last-in, first-out (LIFO) data structure
- Real-world stacks
  - Plate dispensers in the cafeteria
  - Pancakes!
- Some uses:
  - Tracking paths through a maze
  - Providing "unlimited undo" in an application

Operations Provided	Efficiency
Push item	O(1)
Pop item	O(1)

Implemented by `Stack`, `LinkedList`, and `ArrayDeque` in Java

Q1

Queues

- A first-in, first-out (FIFO) data structure
- Real-world queues
  - Waiting line at the BMV
  - Character on Star Trek TNG
- Some uses:
  - Scheduling access to shared resource (e.g., printer)

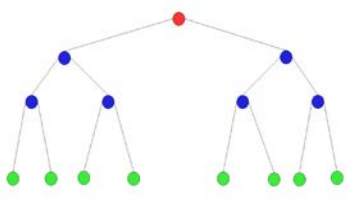
Operations Provided	Efficiency
Add (enqueue, offer) item	O(1)
Remove (dequeue, poll) item	O(1)

Implemented by **`LinkedList`** and **`ArrayDeque`** in Java

Q2

When using a set or map, you choose the implementation:

- Use if you need the items to be **sorted**
- Log(n) height of tree



Binary Tree

- Uses “hash code”
- O(1) to lookup, add or remove

Hash Table

Sets

- Collections **without duplicates**
- Real-world sets
  - Students
  - Collectibles
- Some uses:
  - Quickly checking if an item is in a collection
- Sorted? Depends on implementation!

Operations	HashSet	TreeSet
Add/remove item	O(1)	O(log n)
Contains?	O(1)	O(log n)

Can hog space

Sorts items!

Q3

Maps

- Associate keys with values
- Real-world "maps"
  - Dictionary
  - Phone book
- Some uses:
  - Associating student ID with transcript
  - Associating name with high scores

Operations	HashMap	TreeMap
Insert key-value pair	O(1)	O(lg n)
Look up value for key	O(1)	O(lg n)

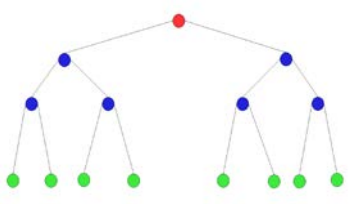
Can hog space

Sorts items by key!

Q4

When using a set or map, you choose the implementation:

- Use if you need the items to be **sorted**
- Log(n) height of tree



Binary Tree

- Uses “hash code”
- O(1) to lookup, add or remove

Hash Table

Generic Types

- Another way to make code more re-useful

Before Generics…

- Java Collections just stored **Objects**
  - This was better than creating different collection classes for each kind of object to be stored
  - Could put anything in them because of **polymorphism**

- Used class casts to get the types right:
  - ```
    ArrayList songs = new ArrayList();
    songs.add(new Song("Dawn Chorus", "Modern English"));
    …
    Song s = (Song) songs.get(1);
    ```
  - ```
    songs.add(new Artist("A Flock of Seagulls"));
    Song t = (Song) songs.get(2);
    ```

run-time error

Q9

With Generics…

- Can define collections and other classes using type parameters
  - ```
    ArrayList<Song> songs = new ArrayList<Song>();
    songs.add(new Song("Dawn Chorus", "Modern English"));
    …
    Song s = songs.get(1); // no cast needed
    ```
  - ~~`songs.add(new Artist("A Flock of Seagulls"));`~~

compile-time error

- Lets us use these classes:
  - in a variety of circumstances
  - with strong type checking
  - without having to write lots of casts

Q10
    ```

Example

- Create a doubly linked list
- Include `min()` and `max()` methods
- Use **polymorphism** rather than null checks for the start and end of the list
- Include `fromArray()` factory method

Q11-Q13

Generics Recap

- Type parameters:
  - `class DLList<E>`
- Bounds:
  - `class DLList<E extends Comparable>`
  - `class DLList<E extends Comparable<E>>`
  - `class DLList<E extends Comparable<? super E>>`
- Generic methods:
  - `public static <T> void shuffle(T[ ] array)`
- http://docs.oracle.com/javase/tutorial/java/generics/index.html

Q14–15, turn in

Project demo/presentation Wednesday

- Business casual
- Think of it as an internal company presentation, not a presentation to the public
- Five-minute presentation, two minutes for questions, two minutes for transition to next team
- Order of teams will be randomly determined

Project demo/presentation Wednesday

- Do a *quick* demo of your project
  - Show off any "extra" features or things that work well
- What part was your team's biggest challenge?
- Show one or two interesting code snippets
  - Highlight your good OO design
- Ask for questions
  - **And ask questions of other teams**

- Before Wednesday, practice getting your computer working with a New Olin projector
  - Remember maximum resolution

Final Exam

- Exam is Wednesday, May 22 at 1:00 pm
- Same general format as previous exams
- Same resources:
  - Paper part: 1 page of notes
  - Computer part: Open book, notes, computer; course web pages and ANGEL pages, JDK documentation, programs in YOUR CSSE220 repositories
- Comprehensive, but focused on Ch 9-18
- May include problems that make sure you understand your team's project code

Final Exam – possible topics

- Simple recursion
- Mutual recursion
- Time-space trade-offs
- Basic search algorithms
  - Binary search, linear search
  - Efficiency, best/worst case inputs
- Big-oh notation, estimating big-oh behavior of code
- File I/O, exception handling
- Function objects
- Linked-list implementation
- Basic data structure use and efficiency
  - ArrayList, LinkedList, Stack, Queue, HashSet, TreeSet, HashMap, TreeMap
- Multithreading (not locks)

Final Exam – possible topics

- Interfaces, polymorphism, inheritance and abstract classes
- Exception handling (try, catch, finally, throw, throws)
- OO design and UML class diagrams
- Basic sorting algorithm
  - Insertion sort
  - Selection sort
  - Merge sort
  - Big-oh analysis of each
- Generic programming
- Event handling, layout managers, exploring the Swing documentation
- Your LodeRunner implementation

Course Evaluations

- Your chance to improve instruction, courses, and curricula.

LodeRunner Work Time