CS310 – Advanced Data Structures and Algorithms

Class 25: Final Review
Midterm Review

• Hopefully worth rereading/viewing
• Highlights:
  • APIs and encapsulation: crucial concepts
    • Emphasized in all 4 PAs
  • Java Interfaces: help with API definition and bridging across multiple implementations of the same API
    • HW4#2 interface facts, Pas 1,2,3
  • Big-Oh expressions: mostly used for CPU-time performance of data structures

5/10/2021
Iteration

- Recall Collection interface: for Set, List, Stack, Queue, PriorityQueue of JDK, but Map has its own API
- Collection implements Iterable
- Iterable<T> interface: has just one method “Iterator<T> iterator()”, so all Collection classes have this method. Note lots of S&W collections are Iterable<T>, such as Graph’s adj, and DijkstraSP’s pathTo.
- Iterator interface: boolean hasNext, T next(), [remove()] (remove is never used in S&W)
- Extra feature: if a collection’s class is Iterable<T>, we can use the enhanced for loop with it: for (T e: collection)
Midterm Review coverage

• Map apps and Set apps, various examples
  • HW4#4 Set app on roommate matching
• Hash Tables
• Inner classes
• Sorting
  • HW4 #1 JDK sorting
• Divide and conquer algorithms
Topics since those on midterm review

- Priority Queues
- Introduced in original tour of Collections classes, at end of JDK specialized Lists slide set
  - insert(item) insert into proper place in PQ
  - FindMin() return min element by priority
  - DeleteMin() delete and return min element by priority
  - isEmpty() and/or size() also needed
Priority Queue apps

- Event-driven Simulation uses PQ (class of Feb. 25)
- An event happening spawns one or more future events, priority = time in future
- Min priority = next event to happen
- You don’t need to follow all the code, just this idea
- Recently, we’ve see the PQ at work in graph algorithms, using S&W PQ classes, including the “souped up” PQ IndexMinPQ with changeKey.
Inheritance

• Class coverage: class of March 9
• PA2: experience coding in a subclass HashMap of a superclass AbstractMap that implements most methods.
  • Shows use of inheritance to avoid code duplication in JDK
• HW 4 #1: A Circle ISA Shape by inheritance, that is, Circle is a subclass of Shape.
  • Shape has abstract methods area and perimeter, which work like an interface
  • Shape also has semiperimeter() and compareTo with code, unlike an interface
Greedy Algorithms

- Intro: Class coverage: **March 4** Change making (US coinage), Interval scheduling, **March 9** image processing
  - HW4 #2 Greedy algorithm applied to non-US currency, yielding suboptimal result (then use DP for good result)
- Huffman’s Algorithm
  - Text: Sec. 5.5, Class coverage: **Apr. 15**, HW5 #1
- Graphs: Dijkstra’s algorithm, (also MST algorithms, but not for final exam)
  - Class coverage: **May 4**, PA4 step 4
Algorithms: Dynamic Programming (DP)

- Intro: change with non-US coinage
  - Class coverage: March 4 Change making (non-US coinage), with detailed example
    - HW4 #2b, like detailed example

- Games: DP for minimax search
  - Class coverage: Apr. 6
  - PA3: provided for TicTacToe, part 4b is DP for Nim
Games

• Class coverage: Apr. 1, Apr. 6, pa3, hw4 #3, #4, #5, pa3
• Concept of game tree
• One play vs. two-player games
• Minimax recursive search for best move for two-player games (vs. simpler search for one-player games)
• Use of DP in that minimax search.

5/10/2021
Data Compression

- Text: Sec. 5.5
- Class coverage: Apr. 13, Apr. 18
- Expressing binary data in 0s and 1s, hex digits
- Using fixed-length binary codes for compression HW5#2
- Huffman’s algorithm for variable-length binary codes HW5#1
Graphs

- Text: Sec. 4.1 undirected graphs, 4.2 directed graphs, 4.4 to Dijkstra’s algorithm
- Class coverage: Apr. 22, 29, May 4, 6 (last is optional)
- HW5 #3, 4, 5, 6, 7, 8, 9, pa4
The projects

• The projects are crucial, and will be covered on the final exam, including possibly pa1 and pa2.
• If you didn’t have time to complete one, be sure to read the posted solution. (pa4’s coming Sunday)
• See the review slides for pa3 and pa4.
Project Terminology: classes and their clients

- Pa1: Xref is client to Tokenizer, with JavaTokenizer interface between.
- PA2: TestMap and TestMapPerf are clients of HashMapx, with Map interface between.
  - In this case, we had the clients in a different package from the classes they used, making sure that they could only use the public interface.
Project Terminology: classes and their clients

- PA3: PlayTicTacToe is client to TicTacToe, and PlayNim is client to Nim, then PlayGame is client to either game class with Game interface between

- PA4: MetroSystem, StationGraph, and PlatformGraph as provided didn’t have a client class, only some client-like code in their main methods. Together they provide the MetroSystem APIs: the MetroSystem API, the StationGraph API, and the PlatformGraph API.
  - Meaning that code could be put in another class as a proper client class.
  - In ShortestPath and ShortestWPath, we have proper client code to the MetroSystem APIs.
Project Goal: proper encapsulation

• When class A uses class B (by calling its methods), so class A is a client of class B, that means class B, the provider of services to A, should be particularly well engineered.

• Proper encapsulation of these service classes is very important: the Tokenizers in pa1, the HashMap in pa2, the game classes in pa3, the MetroSytem and its *Graph in pa4. Less important for the client classes.

• So pa2 and pa3 started out by sealing up their service classes. To seal up a class:
  • You need to determine its public API, make those methods public
  • You need to make all other methods, all inner classes, and of course all fields (instance variables) private.

5/10/2021
Project Theme: interfaces

- Interfaces for APIs were used in pa1, pa2, and pa3, but not pa4 (pa4 does use the JDK-defined Iterable interface).
- In pa3, we saw how multiple implementation classes can be ganged together under one umbrella API, so clients can use one or the other as desired.
- In pa3, the clients got different services (TicTacToe or Nim) that could be run through a common API. In fact many other games could be handled by this API.
Interfaces and pa2

- Although not explored in pa2 itself, we know that the interface in use there, Map, is very important to JDK clients.
- Both HashMap and TreeMap implement Map, so we can switch an app from using HashMap to using TreeMap quite easily:
  - Change the creation of the Map object
  - Use Map as the type for the map variable.
  - Make sure the domain elements implement Comparable or provide a Comparator in the constructor call (now easy to create in Java 8+)
Project Theme: using APIs

- A Java interface helps to describe an API, but it can’t have constructors, and they are also important.
- Any class has an API, although we don’t usually use the terminology for classes that are obviously only client classes.
- The API = the headers of public methods (other than main) and public constructors. Public constants are also allowed here.
- So the API lists the things that the client can do with the class, even if the client class is in another package.

5/10/2021
Using APIs ➔ Inventing APIs

• Hopefully your experience working with all these predefined APIs has shown you how it works.
• The next step is inventing your own APIs.
• Subdividing the work between client and service classes, or between one service class and another.
• This is covered in cs410, Software Engineering.
• Happy Inventing!
Teacher Evaluations

• On Tuesday, May 11, our last class day, I’ll email you a link to use using UMB email, before class.
• Please fill out the linked form (specific to this class) as soon as possible, hopefully in class Tuesday.
• As usual, I won’t see these evaluations until grades have been submitted.