Review for FPP Final
Key Information

The final exam will be a paper exam, without notes or access to a computer. It will cover Lessons 8 – 12. It will consist of information questions and programming questions.

1. **Information Questions (True/False, Multiple Choice, Matching, Short Answer)** These will draw upon the following list of topics from Lessons 8-12**.**

   a. Be familiar with the different advantages and disadvantages of the different ADT's (Abstract Data Types) and implementations we have discussed in class: ArrayList, LinkedList, BST, Hashtable, Set.

      Highlights of 'Lesson 11 - Java Data Structures.pdf" in lectureSlides folder.

      ArrayList – good for randomly accessing elements – i.e. reading values by index. Not good for many insertions and deletions

      LinkedList – good for insertions and deletions; but not good for searching by index.

      BST – it keeps data in sorted order efficiently: fast insertion, deletion, and searching.

      Hashtable – almost random access for looking up by key (and keys don't need to be integers as they do for arrays); elements are not stored in any particuar order (so not good for maintaining sorted order)

      Set – eliminates duplicates automatically; has no inherent order unless you are using TreeSet (TreeSet is just a BST with operations for a Set like add, remove, find)

   b. Understand how each of the following types of data structures are designed and implemented (in a general way): array lists, linked lists, stacks, queues hashtables, bsts, hashsets.

      See the descriptions in the 'Lesson 11 - Java Data Structures.pdf" file in Sakai under the lectureSlides folder.

   c. Understand the ordering principle of a stack (LIFO) and how it is used for something like symbol balancing.

      Be able to determine if a line of code has balanced symbols or not.

   d. Know the classes and interfaces involved in creating a user-defined Collection (like a List) in a way that can make use of Collections sorting and searching methods. Review when the List and Random access interfaces are (or should be) implemented. Be familiar with what you need to do to a list that you create so that it can work with the sort and search functions available in the Collections class.

Lesson 8, Slides 47 and 48

If you want to… with your list….	You must …
Sort using Collections.sort(mylist)	MyList must either implement List directly or be a subclass of AbstractList
Perform fast binary search on MyList when it contains already sorted elements using Collections.binarySearch(mylist)	MyList must implement RandomAccess interface and implement List (or extend AbstractList). If RandomAccess not implemented, binarySearch method will search by doing an ordinary scan

e. Understand the difference between errors, unchecked exceptions, checked exceptions, and runtime exceptions. Know the most common examples of each type.

top level: Throwable
next level: Exception, Error
next: RuntimeException

Examples of Error: StackOverflowError, ClassNotFoundError, ThreadDeath
Examples of RuntimeException: NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException, NumberFormatException, IllegalArgumentException (don't do try/catch)

Exception (checked exceptions): IOException, SQLException, FileNotFoundException, ClassCastException

f. Understand the finally keyword and be able to think through the behavior of a code sample like the one given at the end of the lesson on Exceptions. (A practice quiz is available in the exam preparation folder: Exam Prep Materials, Final called practciceQuizExceptions.pdf.)

g. Know which background data structure is typically used to implement Array Lists, Linked Lists, Binary Search Trees, Hashtables, and Sets. Another way of saying this same thing is: Many data structures are implemented using the composition pattern, relying on some kind of background data structure to perform its operations. If given several data structures you may need to match the data structure to the type of structure most often used as its background data structure.

Data Structure	Based on …
ArrayList	Array

LinkedList	Node
BST	Node
HashMap	Array, LinkedList, Entry
Stack	Could use Array or Node or LinkedList
Queue	Could use Array or Node or LinkedList
TreeSet	Binary Search Tree

h. Understand how the hashCode function is used in a class to support the use of objects as keys in a hashtable.

- Understand how a hashtable transforms input keys to hashcodes then to hashvalues:

  key -> hashCode -> hashvalue = index into the underlying table (array)

  hashCode for key is gotten by calling key.hashCode()
  Minor: hashvalue is computed from a hashcode by using a formula like:
  Math.abs(hashcode % tablesize)

- Know why equal objects must have equal hashCodes and why it is *desirable* for unequal objects to have different hashCodes. Be familiar with best practices concerning the creation of a hashCode. Understand what this means with code. Details from conceptual perspective:

  - Better to have hashcodes be transformed to different hashvalues, which go to different indices in the hashtable. If unequal objects have the same hashCodes, then each list in the table slots becomes very long and lookups take about as long as searching an ordinary linked list. Different hashCodes for unequal objects is better for distributing the objects across more slots in the hashTable. Note: Using a constant value for the hashcode creates an unbalanced table that is less efficient than using a LinkedList.

  - Equal objects must have equal hashCodes so that the hashCode will always access the correct value in the hashTable. (When the key is an object other than String, then the default hashCode will not provide a hashvalue that gives an equal key every time.) If equal objects have different hashcodes, when one key object is placed in the table, it will not be possible to find it using another instance of that key which is equal to it.

i. Given a sequence of ordered values (like integers or Strings), be able to follow the insertion rules to insert them into an initially empty BST.

Check the BST slides (Lesson 10) for an example and lab 10 pencil

2. **Programming Questions:**

   j. Be able to write code for a LinkedList implementation. Be sure that you can use Nodes to implement a method of one of the data structures. (Lab 9 1 and 9-4).

   k. Know Java's Implementation of a LinkedList.

      a. In slide 24 of lesson 9, it mentions that the Java library LinkedList has a method to add elements to the LinkedList:

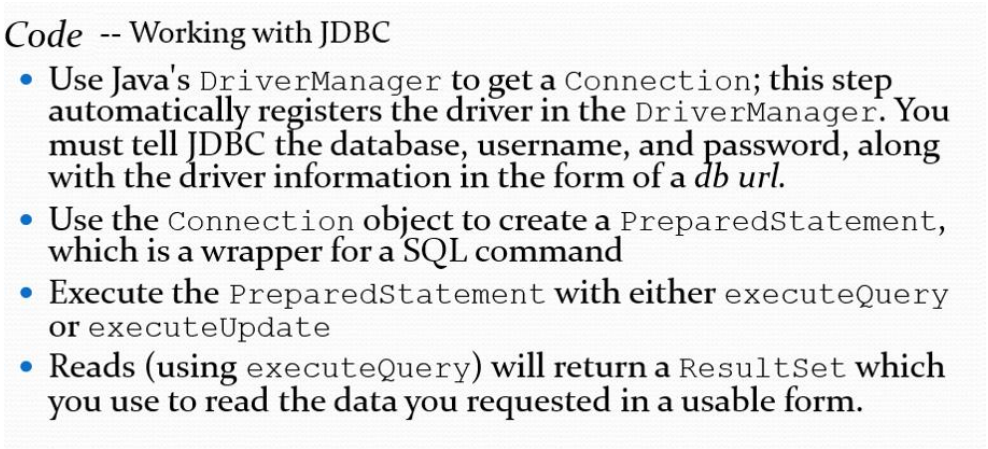
         - void **add**(E obj) - same as enqueue

         You can also use the get(0) operation mentioned in the LectureSlides of Lesson 9. However, one of the problems will be easier if you become familiar with one of the following methods in a LinkedList: `contains` or `indexOf` method in the JDK for the LinkedList.

         public boolean **contains**(Object o).
         Returns true if this list contains the specified element. More formally, returns true if and only if this list of Lesson contains at least one element e such that Objects.equals(o, e).

   l. Be able to write code using a HashMap. (Lab 11-1 and 11-2)

3. **Not Covered on the Final -** The 4 Steps for creating a db connection and executing a query using JDBC (listed in the slides).



   a. Application of the Iterable interface
   b. File I/O
   c. lambdas
   d. JDBC coding