Review for FPP Final (Solutions)

The final exam will be a paper exam, without notes or access to a computer. It will cover Lessons 7 – 13. It will consist of information questions (50%) and programming questions (50%).

1. **Information Questions (T/F, Multiple Choice, Short Answer).** These will draw upon the following list of topics from Lessons 7-13.

   a. *Recursion.* Be familiar with the following points:
      - What has to be true for a recursion to be a *valid* recursion?
        i. Must have a base case
        ii. Sequence of self-calls must lead to the base case
      - Be able to implement (in code) a recursive strategy to solve a problem (as in Lab 7). (Look over solutions to Lab 7)
      - Know what it means to say that a recursive method is inefficient because it performs redundant computations, and what can be done to avoid this situation.

        For example: fib(n) had lots of redundant computations (for instance fib(2) is computed many times). Redundant computation take time (are costly) so should be avoided. One way to avoid is: do an iterative implementation (like with a for loop)

   b. Be familiar with the different advantages and disadvantages of the different ADT's and implementations we have discussed in class: ArrayList, LinkedList, BST, Hashtable, Set.

      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)

c. 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 and the reasons for using the class AbstractList. 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.

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
Use a forEach construct like for(Object ob : mylist)	MyList must implement Iterable (which has as its only method iterator(), which returns an instance of Iterator)
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

d. Know how to use an Iterator object to iterate through the elements of a list. Know the difference between the Iterable and Iterator interfaces. Know the role of Iterator in the use of the for each construct.

```
List myList = Arrays.asList("A", "C", "B");
Iterator it = myList.iterator();
while(it.hasNext()) {
    System.out.println(it.next());
}

interface Iterable {
     Iterator iterator();
     default void forEach
}
interface Iterator<T> {
      boolean hasNext();
      T next();
}
```

e. Be familiar with the top level of the exceptions hierarchy provided in Java. 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
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 Lesson 12. (Look at final slides in Lesson 12.)

g. Know which background data structure is typically used to implement Array Lists, Linked Lists, Binary Search Trees, Hashtables, and Sets.

Data Structure	Based on …
ArrayList	array
LinkedList	Node
BST	Node
HashMap	Array, LinkedList (or Node)
Stack	Could use array or Node or LinkedList
Queue	Could use array or Node or LinkedList
TreeSet	BST

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 to hashvalues (and know the difference between these). 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. Make sure you can write code to override hashCode in any class that you create.

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

hashCode for key is gotten by calling key.hashCode()

hashvalue is computed from a hashcode by using a formula like:
Math.abs(hashcode % tablesize)

a. necessary that equal objects have equal hashCodes because: if equal objects have different hashcodes, when one object is placed in the table, it will not be possible to find it using another instance that is equal to it
b. advisable to have unequal objects have different hashCodes because if there are not enough different table slots in use, then each list in the table slots becomes very long and lookups take about as long as searching an ordinary linked list

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 a worked example

j. Be able to write the code for a LinkedList implementation. In particular, be sure that you are familiar with the technique of iterating from a top node (like a header) to some target node in the list. You may be asked to use Nodes to implement other data structures.

Pay attention to solutions to prog 8-2 (lots of sample code in lecture code for lesson 8)

k. Be able to load a BST by hand using an insertion sequence (sample in Lesson 10)

l. 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. Note that stacks and queues can be implemented in more than one way.

m. Be able to explain why, for ordering objects, sometimes the Comparable interface is not enough.

If you need to sort in different ways, Comparable doesn't work because it commits you to one sorting strategy (by name, by salary, etc). Instead you can use Comparator – can have a NameComparator, SalaryComparator.

n. Be able to explain what it means for a Comparator to be consistent with equals, and why, generally speaking, Comparators *should be* consistent with equals. Be able to create your own Comparator so that it is consistent with equals and be able to use it in a call to Collections.sort.

comparator is consistent with equals if the output of compare (ob1, ob2) is 0 only if ob1.equals(ob2) is true; to ensure a Comparator is consistent with equals, the compare method must take into account same variables as equals

o. Know the 4 Steps for creating a db connection and executing a query using JDBC (listed in the slides).

Code -- Working with JDBC

- Use Java's `DriverManager` to get a `Connection`; this step automatically registers the driver in the `DriverManager`. You must tell JDBC the database, username, and password, along with the driver information in the form of a *db url*.
- Use the `Connection` object to create a `PreparedStatement`, which is a wrapper for a SQL command
- Execute the `PreparedStatement` with either `executeQuery` or `executeUpdate`
- Reads (using `executeQuery`) will return a `ResultSet` which you use to read the data you requested in a usable form.

2. **Not Covered on the Final**
    a. Application of the new forEach method in the Iterable interface (you need to know that it is there, but you will not have to write code that uses it)
    b. File I/O
    c. lambdas
    d. JDBC coding (you need to know the steps, as listed above, but you will not write code that accesses a database)

3. **Programming Questions.** There will be two or three programming questions:

    a. *Implement a data structure*. You will be given the type of background data structure to use; you will use that background data structure to implement some or all of the main operations of the main data structure.

    b. *Solve a problem using recursion*. This problem will be similar to problems you did in Lab 7.

    c. *Introduce your own Exception class.* This problem will be similar to the exercises in Lab 12.

4. **SCI.** You will be asked to write a short essay to explain how one or more SCI principles are displayed in the realm of computer science. Richer content will be awarded more credit.