Midterm Topics:

1. Data Structures: know the typical fields/methods for each data structure and pros/cons of each one. Be able to use them in an application.
   - Arrays (Review: Class 2-3, HW 1 & 4)
   - Linked Lists (Review: Class 7-8, Lab 4, HW 4 & 5)
   - Stacks (Review: Class 9-10, Lab 5, HW 5)
   - Queues (Lab 6, Class 12)
2. Java fundamentals
   - Class structure (fields, methods, constructors)
   - Java keywords and vocab (Review: vocab list)
   - Primitive types vs. objects
3. Inheritance and Interfaces (Review: Class 4-5)
   - Syntax and keywords
   - Difference between implementing an interface and extending a class
   - Uses in Graphics and GUIs.
4. Runtime analysis (Review: Class 7-8, HW 4)
   - Relationship between runtime analysis and loops
   - Using "compareTo" operations to evaluate sorting runtimes
   - Big O notation, i.e. $O(1), O(n), O(n^2)$, etc.
5. Algorithms
   - Sorting: insertion sort and bubble sort (Review: Class 8, HW 4)
   - Stack algorithms: arithmetic expressions (Review: Class 10, HW 5)
   - Using a queue to handle different speed of input/output (Review: Lab 6)
6. Misc
   - Iterators (Review: Video, HW 5)
   - Generics (Review: Class 9, last few HWs and Labs)

Midterm Practice:

1. Loops, Lists, Arrays, and Generics. Suppose there is a House class, that implements Comparable<House>. It has one private field, an int, representing the price of the house. There is a constructor that creates a new House with a given initial price. A House is greater than another House if it has a higher price. 212 has been tasked with writing two methods in main: one to find the House with the minimum price given a list of Houses, and one to find the index of a House with a given price in a given array of Houses. Student X has written the following code to accomplish these tasks.

```
/** Return the house with the minimum price. */
public static House minHouse(LinkedList<House> list) {
    House minHouse = list.get(0); // start out with the first house
    int i=1;
    while (i < list.size()) {
        if (list.get(i).compareTo(minHouse) < 0) {
            minHouse = list.get(i);
        }
        i++;
    }
    return minHouse;
}

/**
 * Return the index of the house with the given price.
 * If no house with this price, return -1.
 */
public static int getPriceIndex(House[] array, int price) {
    int rightIndex = -1;
    for (int i=0; i < array.length; i++) {
        if (array[i].getPrice() == price) {
            rightIndex = i;
        }
    }
    return rightIndex;
}
```

Do these functions accomplish the desired tasks? Why or why not? If not, correct the method(s). If yes, find a way to make the code faster or more elegant. If you have time, write the House class (make sure to include a constructor and all necessary methods).

2. Queues and Runtime. 212 is given a Queue class with the following completely functional methods: add, remove, isEmpty, element, and size. Then the task is to write a method within the Queue class that adds all the elements of another Queue, B, to the end of the given queue (while still maintaining order within B). However, this must be accomplished without destroying B (note we shouldn't use uppercase, one-letter variable names, but it is easier to write on paper with shorter variables). Student X has written the following code.

```
private void addAll(Queue<E> B) {
    while (!B.isEmpty()) {
        E element = B.remove();
        this.add(element);
        B.add(element);
    }
}
```

What is the error in this code? How could you fix it? How could you fix it if you didn't have the size method? What is the runtime of your resulting method?

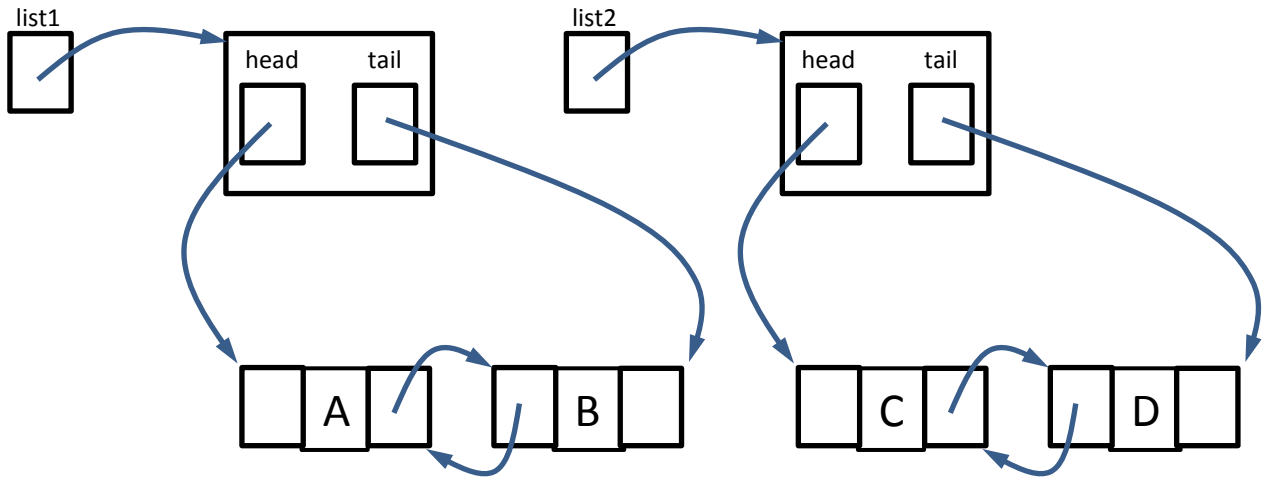
3. Iterators. Use an iterator to write a few lines of code (not a method) to split a linked list of Strings into two linked lists of Strings, with every other element going to a different list. You may assume the original list has an even number of elements.

4. Data Structures (Fall 15).

```
import java.util.*;
public class DataStructure<E> {
    private Stack<E> A;
    private Stack<E> B;

    public DataStructure () {
       A = new Stack<E>();
       B = new Stack<E>();
    }

    private void to(int i) {
       for (int j = 0; j < i; j++) {
           B.push(A.pop());
       }
    }

    private void from(int i) {
       for (int j = 0; j < i; j++) {
           A.push(B.pop());
       }
    }

    public E get(int i) {
       E answer;
       to(i);
       answer = A.pop();
       A.push(answer);
       from(i);
       return answer;
    }

    public void set(E x, int i) {
       to(i);
       if (!A.isEmpty()) {
           A.pop();
       }
       A.push(x);
       from(i);
    }
}
```

a.) Which methods of this class may be called by a different class?
b.) Which abstract data structure is this class implementing?
c.) How does this implementation compare to the usual implementation in efficiency?
d.) Looking just at the public methods of this class, is there any way for a programmer to tell that it is implemented using stacks? Explain.

5. Linked Lists (Fall 15).



a.) Write code to update the links as necessary to remove B from list1.

c.) Write code to swap the order of the elements in list1. Draw a diagram showing the link structure after this operation.