CS 61B Discussion 6 Spring 2017

1 Immutable Rocks

A class is immutable if nothing about its instances can change after they are constructed. Which of the following classes are immutable?

```java
public class Pebble {
    public int weight;
    public Pebble() { weight = 1; }
}
public class Rock {
    public final int weight;
    public Rock (int w) { weight = w; }
}
public class Rocks {
    public final Rock[] rocks;
    public Rocks (Rock[] rox) { rocks = rox; }
}
public class SecretRocks {
    private Rock[] rocks;
    public SecretRocks(Rock[] rox) { rocks = rox; }
}
public class PunkRock {
    private final Rock[] band;
    public PunkRock (Rock yRoad) { band = {yRoad}; }
    public Rock[] myBand() {
        return band;
    }
}
public class MommaRock {
    public static final Pebble baby = new Pebble();
}
```

2 Assorted ADTs

Below are some sketches of ADTs (not real Java code). It's not important to understand the details of how these work right now; just try to understand how each one can be used conceptually.

```
List {
    insert(item, position);       // inserts item into the list at the position
    get(position);              // returns the item in the list at the position
    size();                          // returns the number of items in the list
}

Set {
    add(item);                  // puts item in the set. Does not add duplicates
    contains(item);             // returns whether or not the item is in the set
    items();              // returns a List of all items in some arbitrary order
}

Stack {
    push(item);                                      // puts item onto the stack
    pop();                      // removes and returns the most recently put item
    isEmpty();                               // returns whether the stack is empty
}

Queue {
    enqueue(item);                                   // puts item into the queue
    dequeue();                 // removes and returns the least recently put item
    isEmpty();                               // returns whether the queue is empty
}

PriorityQueue {
    enqueue(item, priority);       // puts item into the queue with a priority
    dequeue();               // removes and returns the item with highest priority
    peek();       // returns but does not remove the item with highest priority
}

Map {                                          // like a dictionary from python
    put(key, value);        /* puts key into the map and associates it with the
                            given value. If key is already in the map, replaces
                                     its existing value with the given value */
    get(key);                               // returns value associated with key
    keys();                // returns a List of all keys in some arbitrary order
}
```

3 Solving Problems with ADTs

Consider the problems below. Which of the ADTs given in the previous section might you use to solve each problem? Although in principle any of the ADTs might be used to solve any of the problems, think about which ones will make code implementation easier or more efficient.

1. Given a news article, find the frequency of each word used in the article.

2. Given an unsorted array of integers, return the array sorted from least to greatest.

3. Implement the forward and back buttons for a web browser.

4 Design a Parking Lot!

Design a `ParkingLot` package that allocates specific parking spaces to cars in a smart way. Decide what classes you'll need, and design the API for each. Time permitting, select data structures to implement the API for each class. Try to deal with annoying cases (like disobedient humans).

- Parking spaces can be either regular, compact, or handicapped-only.
- When a new car arrives, the system should assign a specific space based on the type of car.
- All cars are allowed to park in regular spots. Thus, compact cars can park in both compact spaces and regular spaces.
- When a car leaves, the system should record that the space is free.
- Your package should be designed in a manner that allows different parking lots to have different numbers of spaces for each of the 3 types.
- Parking spots should have a sense of closeness to the entrance. When parking a car, place it as close to the entrance as possible. Assume these distances are distinct.