Midterm 1: Compsci 201

Tabitha Peck

October 2, 2013

Print your name and NetID legibly in ALL CAPITAL letters. Make sure that we can clearly determine L vs. 1 and S vs. 5. It will effect your grade if you do not follow these instructions or we cannot read your name or netID.

Name: ___________________________________________

NetID/Login: _________

Honor code acknowledgment (signature) ____________________________

This test has 15 pages (with a help page and APT at the end), be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.

In writing code you do not need to worry about specifying the proper import statements. Don’t worry about getting function or method names exactly right. Assume that all libraries and packages are imported in any code you write.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>18 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>14 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 6</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL:</strong></td>
<td>80 pts.</td>
<td></td>
</tr>
</tbody>
</table>
You are to implement a class `Student` to satisfy several criteria. You'll implement a constructor, you'll override `.equals` and `.hashCode`, and you'll implement `compareTo` since `Student` implements the `Comparable` interface.

Each `Student` object stores a student’s name (String), a student’s courage (int), and a student’s determination (int). A code example of how `Student` objects are used is shown below. For those of you who have read or seen any *Harry Potter* books or movies you can think of the `Student` class as facilitating sorting as done by the Sorting Hat to determine which student wizards are placed in Gryffindor house. But knowledge of Harry Potter isn’t necessary to understand this question.

- Two `Student` objects are equal if they have the same name, the same courage value, and the same determination value.

- You should be sure your `.hashCode` method is implemented according to best principles of correctness and performance.

- When sorting, courage values are most important, high courage values come first when `Student` objects are sorted. When courage values are equal, higher determination values come first, and if both courage and determination values are equal, `Student` objects should be sorted in alphabetical order.

- Assume that `Student.toString` has been implemented, you don’t need to implement it.

The coding example below shows how `Student` objects should work when constructing and sorting `Student` objects. In the function `sortStudents` parameter `name` is an array of student names, parameter `courage` is an array of student courage values (0 is none, 10 is extremely high), and parameter `determine` is an array of student determination, again on a scale of 0-10. The name at `name[i]` has a courage value given by `courage[i]` and a determination given by `determine[i]`.

**Example:**

Input:

- `name = ["Potter, Harry", "Granger, Hermione", "Malfoy, Draco", "Lovegood, Luna"]`
- `courage = [10, 10, 4, 7]`
- `determine = [10, 10, 9, 8]`

Output:

- "Granger, Hermione", "Potter, Harry", "Lovegood, Luna", "Malfoy, Draco"

**Code:**

```java
public void sortStudents(String[] name, int[] courage, int[] determine){
  ArrayList<Student> list = new ArrayList<Student>();
  for(int i = 0; i < name.length; i++)
    list.add(new Student(name[i], courage[i], determine[i]));
  Collections.sort(list);
  for(Student s: list)
    System.out.println(s.toString());
}
```

*Question continues on the next page.*
Instance variables and constructor: 6 points
Complete the implementation of class Student by adding instance variables and completing the constructor so that the class Student will work with the code given above.

```java
public class Student implements Comparable<Student>{
    // add your instance variable(s) here

    // complete the constructor definition and don’t forget to define parameters

    public Student( ){
    }
}
```

Equals and hash code: 6 points
Complete the hashCode and equals methods below.

```java
public int hashCode(){

}
```

Question continues on the next page.
public boolean equals(Object obj){
    if (obj == this) {
        return true;
    }
    if (obj == null || obj.getClass() != this.getClass()) {
        return false;
    }

    Student temp = (Student) obj;
}

compareTo: 6 points
Complete the compareTo method below.

    public int compareTo(Student arg0) {

}
For each of the following, give the running time in terms of the parameter \( n \) in big-Oh notation. You must justify your answer. (2 points for each running time, and 1 point for each justification)

```
public int numberOne(int n){
    int answer = 1;
    for(int i = 0; i < n*n; i+=2)
        for(int j = 0; j < 2*n; j++)
            answer++;
    return answer;
}
```

Big-Oh: Justification:

```
public int numberTwo(int n){
    int answer = 1;
    for(int i = 0; i < n; i++)
        answer++;
    for(int i = 0; i < 10000; i++)
        for(int j = 0; j < 10000; j++)
            answer++;
    return answer;
}
```

Big-Oh: Justification:

```
public int numberThree(int n){
    int answer = 1;
    for(int i = 1; i <= n; i=i*2)
        answer++;
    return answer;
}
```

Big-Oh: Justification:

```
public int numberFour(int n){
    int answer = 1;
    for(int i = 1; i <= n; i++)
        for(int j = 0; j < i; j++)
            answer++;
    return answer;
}
```

Big-Oh: Justification:
PROBLEM 3 :  *(BB-Queue: 8 points)*

Your friend Sammy really wants to understand queues and has decided to implement a queue from scratch rather than rely on the Java implementation provided in the `java.util` package.

Explain to Sammy how a queue works *(2 points)*

Sammy has decided to use an `ArrayList` for the queue implementation. Explain why using an `ArrayList` may not be the best idea. Make sure to talk about efficiency. *(3 points)*

Give Sammy some advice as to what data structure would be better for implementing a queue. Explain why and make sure to talk about efficiency. *(3 points)*
PROBLEM 4:  (Corny Hash: 14 points)

Sammy wants to write a class, Person, with instance variables String myName and int myAge, the person’s name and age respectively. Sammy has written the following code to override .hashCode and .equals.

```java
public boolean equals(Object obj){
    if (obj == this) {
        return true;
    }
    if (obj == null || obj.getClass() != this.getClass()) {
        return false;
    }
    Person temp = (Person) obj;
    return myName.equals(temp.myName) && myAge == temp.myAge;
}

global int hashCode(){
    return myAge;
}
```

You want to convince Sammy that this may not be the best code. You come up with an example where n Persons, each with different names, but with all with the same age are added to a hash table.

Based on the .equals and .hashCode methods above, what is the running time to determine if the hash table that stores n objects contains a Person object? (3 points)

Running time: ___________ Explain your answer.

Based on the hashCode method above, what is the running time to add n Persons with different names but the same age to an initially empty hash table? (3 points)

Running time: ___________ Explain your answer.
Improve Sammy’s hashCode method by rewriting the code here. (2 points).

Based on your hashCode method, what is the running time to determine if the hash table that stores \( n \) objects contains a Person? object (3 points)

Running time: ______________ Explain your answer.

Based on your hashCode method above, what is the running time to add \( n \) Persons with different names but the same ages to your hash table? (3 points)

Running time: ______________ Explain your answer.
An almost complete solution to the Anonymous APT is below. Complete the method `makeMap` so that the APT tester returns all green! You can find the Anonymous APT writeup at the end of the exam. **Do not modify** any code in `howMany`.

```java
public class Anonymous {
    public int howMany(String[] headlines, String[] messages) {
        int count = 0;
        HashMap<Character, Integer> headmap = new HashMap<Character, Integer>();

        for(String s: headlines){
            makeMap(headmap, s);
        }

        for(String s: messages){
            HashMap<Character, Integer> m = new HashMap<Character, Integer>();
            makeMap(m, s);
            boolean works = true;
            for(Character c: m.keySet()){  
                if(headmap.containsKey(c)){
                    if(m.get(c) > headmap.get(c)){
                        works = false; break;
                    }
                } else{
                    works = false; break;
                }
            }
            if(works)
                count++;
        }
        return count;
    }

    public void makeMap(HashMap<Character, Integer> map, String s){
        }
    }
}
```
PROBLEM 6 : (Linked lists: 20 points)

Complete the code below to implement a Stack using linked lists. Your stack will use the Node class below.

```java
public class Node{
    public Node myNext;
    public int myData;

    public Node(int data, Node next){
        myNext = next;
        myData = data;
    }
}
```

Part A: 4 points

The stack below stores the values (5, 4, 7) where 5 was added first to the stack, then 4, and then 7. The node myTop points to the top of the stack where elements are pushed (added) and popped (removed), and myBottom points to the bottom of the stack.

![Stack Diagram](image)

Draw the stack that results after a pop (remove) is called from the stack diagrammed above. Be sure to include labels for both myBottom and myTop.

Based on your previous drawing, draw the resulting stack after push(2) (add a node with the value 2) is called. Make sure to diagram all relevant pointers.
Part B: 6 points
Below is a partially completed integer stack with both top and bottom pointers, myTop and myBottom. The number of elements in the stack is held in mySize. This code uses the Node class on the previous page.

```java
public class IntStack {
    public Node myTop;
    public Node myBottom;
    public int mySize;

    public IntStack() {
        myTop = null;
        myBottom = null;
        mySize = 0;
    }
    public boolean isEmpty() {
        return mySize == 0;
    }
}

Complete pop.

// Pop a node from the stack. Code has been added for you to throw
// an exception if the stack is empty.

public int pop(){
    if (isEmpty()) throw new java.util.NoSuchElementException();

    // add code here

}
```
Part C: 4 points
Complete push which pushes a node onto a stack

    // push a node onto the stack
    public void push(int data){
    }
}

Part D: 6 points
It is possible to implement a queue using two stacks (and no other containers – no ArrayLists, no arrays). Making calls to push, pop and isEmpty, from above, complete the method dequeue, that removes an element from a queue and returns the value. The method enqueue has been completed for you.

    public class TwoStackQueue extends IntStack{
        private IntStack stackA;
        private IntStack stackB;

        public void enqueue(int i){
            stackA.push(i);
        }

        // removes the first item in the queue and returns its value. 
        // note that if queue is empty an exception is thrown
        public int dequeue(){
            if (stackA.empty()) throw new java.util.NoSuchElementException();
            }
        }
    }
### Problem Statement

If you want to write a message anonymously, one way to do it is to cut out letters from headlines in a newspaper and paste them onto a blank piece of paper to form the message you want to write. Given several headlines that you have cut out, determine how many messages from a list you can write using the letters from the headlines. You should only consider each message by itself and not in conjunction with the others, see example 2.

Write the method `howMany` which takes as parameters a `String[] headlines` containing the headlines which you have cut out as well as a `String[] messages` with the messages you may want to write, and returns an `int` which is the total number of messages you can write.

### Constraints

- All letters that you cut out can be used both as upper or lower case in a message.
- Spaces should be ignored in elements in both headlines and messages.
- `headlines` will contain between 1 and 50 elements, inclusive.
- `messages` will contain between 1 and 50 elements, inclusive.
- The length of each element in `headlines` will be between 1 and 50 characters, inclusive.
- The length of each element in `messages` will be between 1 and 50 characters, inclusive.
- Each element in `headlines` will only contain the letters 'A-Z', 'a-z' and space.
- Each element in `messages` will only contain the letters 'A-Z', 'a-z' and space.
String

- `.length()` Get the length of the `String`. O(1).
- `.charAt(i)` Get the `char` at index `i`. O(1).
- `.split(" ")` Split a string by spaces and store it in a `String[]`.
- `.substring(i, j)` Get the substring between indices `i` and `j`. Index `i` is inclusive, and index `j` is exclusive. O(1). For example:

```java
String x = "abcdefg";
String y = x.substring(2, 4);
// y now has the value "cd"
```

ArrayList<T> // Where T is a type, like String or Integer

- `.add(i, X)` Add element `X` to the list at index `i`. If no `i` is provided, add an element to the end of the list. Adding to the end runs in O(1).
- `.get(i)` Get the element at position `i`. Runs in O(1).
- `.set(i, X)` Set the element at position `i` to the value `X`. O(1).
- `.size()` Get the number of elements. O(1).

HashSet<T> // Where T is a type, like String or Integer

- `.size()` Compute the size. O(1).
- `.add(X)` Add the value `X` to the set. If it’s already in the set, do nothing. O(1).
- `.contains(X)` Return a `boolean` indicating if `X` is in the set. O(1).
- `.remove(X)` Remove `X` from the set. If `X` was not in the set, do nothing. O(1).

HashMap<K, V> // Where K and V are the key and value types, respectively.

- `.size()` Compute the size. O(1).
- `.containsKey(X)` Determines if the map contains a value for the key `X`. To get that value, use `.get()`. O(1).
- `.get(X)` Gets the value for the key `X`. If `X` is not in the map, return `null`. O(1).
- `.put(k, v)` Map the key `k` to the value `v`. If there was already a value for `k`, replace it. O(1).
- `.keySet()` Return a `Set` containing the keys in the map. Useful for iterating over. O(1).

To iterate over a `HashSet<T>`, use

```java
for (T v : nameOfSet) {
    // v is the current element of the set.
}
```

This can be combined with `HashMap`’s `.keySet()` to iterate over a `HashMap`. 