Test 1: CompSci 100 - Exam 1 Solutions

Michael Hewner 2/17/2012

Name: _____Netid (print clearly): _____

Honor code acknowledgement (signature): _____

	value	grader	grade
Question 1: Algorithms 1	5 points		
Question 2: Algorithms 2	5 points		
Question 3: Algorithms 3	5 points		
Question 4: Classes, Equals, HashCode, Comparable 1	3 points		
Question 5: Classes, Equals, HashCode, Comparable 2	2 points		
Question 6: Classes, Equals, HashCode, Comparable 3	5 points		
Question 7: Big O	5 points		
Question 8: Recursion 1	5 points		
Question 9: Recursion 2	5 points		
TOTAL	40 points		

This exam contains 11 pages. Please check to ensure your copy has the right number of pages.

You have 75 minutes to complete this exam. Do NOT spend too much time on any one question. Do not start this exam until your professor gives you permission to do so. When time is called, stop immediately - otherwise your exam will get at 10% penalty.

In writing your code you do not need to worry about import statements.

Question 1: Algorithms 1 [5 points]

Write the function getMostFrequentString that takes a non-empty array of strings and returns the string that occurs most frequently in the list. You can assume that the most frequent string is unique (that is, that no other strings are "tied" for most frequent).

Examples:

{"dog","cat","bird","cat"} returns "cat"
{"dog"} returns "dog"

```
public String getMostFrequentString(String[] inputs) {
```

```
    public String getMostFrequentString(String[] inputs) {
        HashMap<String, Integer> counts = new HashMap<String,Integer>();
        for(String s : inputs) {
            if(!counts.containsKey(s)) {
                counts.put(s,0);
            }
            counts.put(s, counts.get(s) + 1);
        }
        int bestCount = 0;
        String best = null;
        for(String s:counts.keySet()) {
            if(counts.get(s) > bestCount) {
                bestCount = counts.get(s);
                best = s;
            }
        }
        return best;
    }
```

Question 2: Algorithms 2 [5 points]

Imagine that we divided CS100 into a several teams. Each CS100 student belongs to exactly 1 team. We store all these teams in a HashMap. The HashMap maps each team name to the list of students who belong to that team. Write a function getTeamForStudent that takes the hashmap and a student name and returns the team the student belongs to. If the student does not belong to any team, the function should return null.

Example:

myHashMap looks like:

{ "Team Ninja" = ["Steve","Lisa","Siyang"],
"Team Robot" = ["Puneet","Chris","Jen"]}

getTeamForStudent(myHashMap,"Jen") returns "Team Robot"
getTeamForStudent(myHashMap,"Mike") returns null

```
public String getTeamForStudent(HashMap<String,ArrayList<String>> map, String name) {
```

```
    public String getTeamForStudent(HashMap<String,ArrayList<String>> map, String name) {
        for(String team : map.keySet()) {
            for(String student : map.get(team)) {
                if(student.equals(name))
                    return team;
            }
        }
        return null;
    }
```

Question 3: Algorithms 3 [5 points]

Imagine that you're writing software that tracks families. You have a HashMap that maps each person (by name) to the name of their mother. Write a function isFemaleAncestor that takes in the HashMap and two names, child and possibleAncestor. The function should return true if the possibleAncestor is the child's mother, or the child's mother's mother, etc. This question may be solved with recursion, although it is not required.

You can assume the map is well formed (e.g. does not have anybody as their own ancestor). If the child is not a key in the map, isFemaleAncestor should return false. Note that family histories only go back so far, so eventually you may find a mother in the map who's mother is not known (e.g. Eve in the map below).

Example:

myMotherMap looks like:
{"Steve"="Sharon",
"Jane"="Sharon",
"Sharon"="Lisa",
"Lisa"="Eve"}
isFemaleAncestor(myMotherMap , "Steve", "Sharon") returns true
isFemaleAncestor(myMotherMap , "Steve", "Eve") returns true
isFemaleAncestor(myMotherMap , "Steve", "Jane") returns false
isFemaleAncestor(myMotherMap , "Lisa", "Sharon") returns false

```
public boolean isFemaleAncestor(HashMap<String,String> motherMap, String child, String
       possibleAncestor) {
```

```
    public boolean isFemaleAncestor(HashMap<String,String> motherMap, String child,
String possibleAncestor) {
        String current = child;
        while(motherMap.containsKey(current)) {
            String mother = motherMap.get(current);
            if(mother.equals(possibleAncestor)) {
                return true;
            }
            current = mother;
        }
        return false;
    }

    public boolean isFemaleAncestor2(HashMap<String,String> motherMap, String child,
String possibleAncestor) {
        if(!motherMap.containsKey(child))
            return false;
        String mother = motherMap.get(child);
        if(mother.equals(possibleAncestor))
            return true;
        return isFemaleAncestor2(motherMap,mother,possibleAncestor);
    }
```

Question 4: Classes, Equals, HashCode, Comparable 1 [3 points]

```
public interface FunInterface { }

public class ClassA implements FunInterface { }

public abstract class ClassB extends ClassA { public abstract int myFun(); }

public class ClassC extends ClassB {

	public int myFun() { return 8; }

}

public class ClassD extends ClassA { }
```

Circle whether the following lines of code would compile:

`FunInterface var = new ClassA();`	This code would compile	This code would not compile
`ClassA var = new ClassD();`	This code would compile	This code would not compile
`ClassC var = new ClassC();` `System.out.println(var.myFun());`	This code would compile	This code would not compile
`ClassA var = new ClassC();` `System.out.println(var.myFun());`	This code would compile	This code would not compile
`ClassB var = new ClassB();` `System.out.println(var.myFun());`	This code would compile	This code would not compile
`ClassB var = new ClassC();` `System.out.println(var.myFun());`	This code would compile	This code would not compile

Question 5: Classes, Equals, HashCode, Comparable 2 [2 points]

```
public class NumberPlusSmallOffset {
	private int myNum, myOffset;
	public NumberPlusSmallOffset(int num) {
		myNum = num;
		if(num > 200000) {
			//generate a small random offset
			Random r = new Random();
			myOffset = r.nextInt(10);
		} else {
			myOffset = 0;
		}
	}
	public int hashCode() {

		return myNum + 31*myOffset;

	}
	public boolean equals(Object o) {
		if (o == null)
			return false;
		if (getClass() != o.getClass())
			return false;
		NumberPlusSmallOffset other = (NumberPlusSmallOffset) o;
		//offset doesn't matter for equality
		if(other.myNum == myNum) return true;
		return false
	}
}
```

This code violates one of the rules of equals and hashCode, and therefore there are bugs when NumberPlusSmallOffset objects are used as keys in HashMaps. Do two things:

1. Write (in 1 or 2 sentences) what rule the class violates.
2. Then write your version of hashCode (below) to fix the problem.

This class violates the rule that two objects which are equal with .equals must have the same hashcode. Two objects with different offsets may be equal but they will have different hashcodes.

```
	//improved version of hashCode
	public int hashCode() {
		return myNum;

	}
```

Question 6: Classes, Equals, HashCode, Comparable 3 [5 points]

So imagine you're writing a class that represents a team in the regional Paper Scissors Rock competition (PSRTeam). Each PSRTeam is constructed with a team name and an array that indicates which team has won in the various competitions throughout the season. When the teams are sorted, we would like the team who has won the most games to be sorted first, the team who has won the 2nd most games to be second, etc. If 2 teams have the same number of wins, they should be sorted alphabetically by name. Here's some sample code that uses the PSRTeam class you write:

```
String[] competitionWinners = {"Duke","State","UNC","Duke","Duke","UNC"};
ArrayList<PSRTeam> list = new ArrayList<PSRTeam>();
list.add(new PSRTeam("UNC", competitionWinners));
list.add(new PSRTeam("Duke", competitionWinners));
list.add(new PSRTeam("State", competitionWinners));
Collections.sort(list);
//Duke should now be first in the list (index 0) followed by UNC, followed by State
```

Write the code for the PSRTeam class so the above code works:

```
public class PSRTeam implements Comparable<PSRTeam> {
    private String myName;
    private int myNumWins;

    public PSRTeam(String name, String[] wins) {
        myName = name;
        myNumWins = 0;
        for(String winner : wins) {
            if(myName.equals(winner)) {
                myNumWins++;
            }
        }
    }

    public int compareTo(PSRTeam other) {
        if(myNumWins == other.myNumWins)
            return myName.compareTo(other.myName);
        return other.myNumWins - myNumWins;
    }

}
```

Question 7: Big O [5 points]

For each of the following functions, given a specific definition for n, determine the function's runtime as a function of n**. Express your answer using Big O notation as we've discussed in class, omitting coefficients and extra terms.**

1.

```
public long example1(int[] nums)
{
    long sum = 0;
    for(int num1 : nums) {
        for(int num2 : nums) {
            sum += num1*num2;
        }
    }
    return sum;
}
```

Runtime of example1, where n is the number of elements in nums: $O(n^2)$

2.

```
public int example2(ArrayList<Integer> input)
{
    int count = 0;
    for(int i = 0; i < input.size(); i++) {
        for(int j = 0; j < i/3; j++) {
            if(input.get(i) == input.get(j)) {
                count++;
            }
        }
    }
    return count;
}
```

Runtime of example2, where n is the number of elements in input: $O(n^2)$

3.

```
public void example3(ArrayList<Integer> input)
{
    int sum = 0;
    for(int i = 0; i < input.size(); i++) {
        //removes and returns 1st element in the list
        sum += input.remove(0);
    }
}
```

Runtime of example3, where n is the number of elements in input: $O(n^2)$

4.

```
public boolean example4(int input)
{
    int data = input;
    while(data > 1) {
        if(data % 2 != 0)
            return false;
        data = data / 2;
    }
    return true;
}
```

Runtime of example4, where n is the size of input: O(log n)

5.

```
public void example5(String[] strings)
{
    int p = 55;
    if(strings.length < 55) {
        p = strings.length;
    }

    for(int i = 0; i < p; i++) {
        System.out.println("Hello!");
    }

}
```

Runtime of example5, where n is the number of elements in strings: O(1)

Question 8: Recursion 1 [5 points]

You must solve this question using recursion. Your function should contain no loops, and use no variables outside this function.

Write the function hasDoubledLetters that takes a string and returns true if the string consists of pairs of repeated letters.

Examples:

hasDoubledLetters ("ddoogggg") returns true
hasDoubledLetters ("ddogg") returns false (because the o is not doubled)
hasDoubledLetters ("Z") returns false
hasDoubledLetters ("") returns true

```
public boolean hasDoubledLetters(String input) {

    public boolean hasPairedLetters(String input) {
        if(input.length() == 1)
            return false;
        if(input.length() == 0)
            return true;
        if(input.charAt(0) != input.charAt(1))
            return false;
        return hasPairedLetters(input.substring(2));
    }
```

Question 9: Recursion 2 [5 points]

You must solve this question using recursion. Your function should contain no loops, and use no variables outside this function.

A double countdown is a string that counts down to 1 and then counts back up to the original number. Write a function doubleCountdown that given a number outputs a string double countdown with the numbers separated by "…". You can assume that the input number will be 1 or greater.

Examples:

doubleCountdown(3) return "3…2…1…2…3"
doubleCountdown(1) returns "1"

```
public String doubleCountdown(int input) {
```

```
    public String doubleCountdown(int input) {
        if(input == 1)
            return "1";
        return input + "..." + doubleCountdown(input - 1) + "..." + input;
    }
```