CS 180 Problem Solving and OO Programming
Final Exam Practice Problem Set
December 11, 2011

Note: Please make sure you are able to solve all problems given during the recitation sessions without looking at the solutions. *The problems below will be discussed during the special class on Sunday Dec 11, 4pm LWSN 3102.*

1. Write a Java method named GCD that computes the greatest common divisor of two integers `x>=0 and y>=0` using the following definition.

   GCD(x, y) = x, if y=0
   = GCD(y, x%y), otherwise.

2. A single dimensional array `a` is given and is sorted in ascending order. Write a recursive method named `bSearch` that takes array `a` and integer `x` as inputs and finds if `x` is in `a`. A simple recursive binary search procedure is explained below.

(a) If the array is empty then return `false`.
(b) If the array has exactly one element then compare `x` with this element and return `true` if they are the same else return `false`.
(c) If the length of the array is greater than 1 then check if `x` is at the center of `a`, and if so then return `true`.
(d) If `x` is not at the center of `a` but is less than the element at the mid point then search for x among elements to the left of the mid point in a, else search for `x` in the elements to the right of the mid point.

Example:

```
a=[12  14 89 98 102]
x=98
```

Check if `x` is at the center of `a`, i.e., is `x==a[2]`? It is not equal but is greater than 89. Hence search for `x` in the array [98 102]. This time compare `x` with `a[1]`. Again `x` is not equal to `a[1]` but is less than `a[1]`. Hence search in another array `a=[98]`. As the length of this array is 1, simply compare x with `a[0`. In this case `x` is equal to `a[0]` hence return true.

Note that each time `bSearch` is called, it is given a new array as input. Also, you may find the mid point in an array as `int mid=a.length/2;`

3. You are required to write two classes named `Test` and `Sum`. Class `Sum` extends `Thread`. The `main()` method in `Test` creates two objects of type `Sum` named `sEven` and `sOdd`. Both these objects are passed the array as input as well as a string. `sEven` is given the string "even" and `sOdd` is given the string "odd". `main()` then starts the threads `sEven` and `sOdd` and waits for them to complete. Upon completion `main()` gets the sum obtained by each thread and prints them.

   The `Sum` class contains a constructor, a `run()` method and the `getSum()` method. The constructor saves the parameters locally. The `run()` method adds all the elements of the given array that are located at even indices (0, 2, 4,...) if the input string is "even" otherwise adds the elements located at odd indices (1, 3, 5,...). `getSum()` returns the sum computed by the thread.

   Example:

   If the input array is
   ```
   int [] a={3, 4, 8, 90, 12, 0, 2, 3};
   ```

   then the output of `main()` should be:

   Sum of elements at even indices: 25
   Sum of elements at odd indices: 97

4. Modify the `Sum` class from problem 3 so that the `run()` method accesses each element of the input array after a delay of 5 seconds. Thus, for example, if the `run()` method is summing up the elements at even indices, then before accessing a[0], it should wait for 1 second, then access a[0], and then again wait for 1 second before accessing a[2], and so on. **Do not use `Thread.sleep()` to wait.** The `main()` method should print the total time taken for all tasks (from the start of `main()` until the end).

   If the two threads can sum the elements of an array of `n` elements in time `t` (without any delay in accessing the elements) then how much time do you estimate they will take to sum up the even and odd elements if the delay is 5 seconds?

<End of Final Exam Problem Set>