Computer Science I — Fall 2018

Final Exam

This exam should have seven pages, including the page of documentation at the end. Closed book and notes. No calculators or computers allowed. Comments are not required on any code-writing problems

Problem 1: [20 points]

Describe briefly, in plain English, what the mystery method below does. What would be a good, descriptive name for this method?

```java
public static ArrayList<String> mystery(File f) {
    ArrayList<String> x = new ArrayList<String>();
    if (f.exists()) {
        Scanner s = new Scanner(f);
        while(s.hasNext()) {
            String ss = s.nextLine();
            x.add(ss);
        }
    }
    return x;
}
```
Problem 2: [20 points]

a) What characteristic must an array of numbers have in order to use the binary search algorithm on it? Why?

b) What does the final keyword mean when used in defining a variable? Why might you use it in a program?

c) What does the static keyword mean when used in defining a variable? Why might you use it in a program?

d) What do we mean by the "state" of an object?
Problem 3: [40 points]

For the Turtle assignment early in the semester, you implemented a class that kept track of where a virtual “turtle” was on the screen. (The documentation for the class is at the end of the exam.) The next few problems involve writing a class called TurtleMob that maintains a group of Turtle instances, and has methods that operate on the collection.

```
import java.util.ArrayList;

public class TurtleMob
{
    private ArrayList<Turtle> mob; // To hold the Turtles
    // Methods omitted
}
```

a) Please define the constructor for the TurtleMob class. It should take two ints as inputs: The number of Turtle instances desired, and the distance each should travel when they move (all Turtle instances in the group will use the same distance). It should create the specified number of Turtle objects and store them in the list (after creating the list, of course).

b) Define a toString method for the TurtleMob class that returns a string that includes the number of turtles in the mob. Something like "This angry mob contains 37 ferocious turtles", for example, where the actual number of turtles in the mob is used instead of 37.
c) Define a method called `advance` for the `TurtleMob` class that causes each of the turtles in the mob to move forward.

d) Define a method called `adopt` that takes another `TurtleMob` as its input. It should add all of the turtle instances from the other mob to ours. For full credit, make sure the other mob ends up empty when we’re done. (We’re taking their turtles, not cloning them.)
Problem 4: [20 points]

We tested our selection sort code by passing it a few short arrays — short enough that we could manually inspect the results and verify that the method was sorting arrays correctly. It would be nice to automate that testing, and to be able to verify that much longer arrays were sorted correctly as well. Below, define a static method called `ordered`, that takes an array of integers and returns `true` if they’re in order from smallest to largest, `false` otherwise. (Hint: You can check that the values are in order by making a single pass through the array, comparing neighboring items as you go.) For full credit, your solution should handle arrays containing duplicate items properly. You may assume the array contains at least one value.
Problem 5: [20 points]

The selection sort method we wrote in class (and that you worked with in lab) is shown below, with some portions omitted. Fill in the six blanks with the appropriate missing code. (We originally wrote the inner loop as a separate method, but I’ve moved its contents into the body of the sort method so it’s easier to see everything at once.)

```java
public void selectionSort(int[] nums) {
    for(int start=0; start<nums.length-1; start=start+1) {
        int indexOfMin = _____________;
        for(int i=start+1; i<_____________; i=i+1) {
            if (nums[i] < nums[indexOfMin]) {
                indexOfMin = i;
            }
        }
        int temp = nums[______________];
        nums[______________] = nums[______________];
        nums[______________] = temp;
    }
}
```
Class Turtle

This class represents a "turtle" that can be asked to change directions and move around the screen. It's similar to the concept of the Logo turtle, though with only four possible headings.

Constructor Summary

Constructors

<table>
<thead>
<tr>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Turtle()</strong></td>
</tr>
<tr>
<td>This constructor sets up our &quot;turtle&quot; and leaves it facing north.</td>
</tr>
<tr>
<td><strong>Turtle(int initialDistance)</strong></td>
</tr>
<tr>
<td>This second constructor allows the user to specify how far our turtle should move each time forward() is called: It takes a parameter containing this extra information, and uses an assignment statement to store a copy of that value in our distance field.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><strong>faceEast()</strong></td>
</tr>
<tr>
<td></td>
<td>Change the turtle's state so that we're heading east.</td>
</tr>
<tr>
<td>void</td>
<td><strong>faceNorth()</strong></td>
</tr>
<tr>
<td></td>
<td>Change the turtle's state such that we'll head &quot;north&quot; (up the screen) the next time forward() is called.</td>
</tr>
<tr>
<td>void</td>
<td><strong>faceSouth()</strong></td>
</tr>
<tr>
<td></td>
<td>Change the turtle's state so that we're heading south.</td>
</tr>
<tr>
<td>void</td>
<td><strong>faceWest()</strong></td>
</tr>
<tr>
<td></td>
<td>Change the turtle's state so that we're heading west.</td>
</tr>
<tr>
<td>void</td>
<td><strong>forward()</strong></td>
</tr>
<tr>
<td></td>
<td>Move in the direction we're heading, by the number of pixels stored in the distance field.</td>
</tr>
<tr>
<td>void</td>
<td><strong>setDistance(int newDistance)</strong></td>
</tr>
<tr>
<td></td>
<td>The setDistance method can be used to change how far the turtle moves each time forward() is called.</td>
</tr>
</tbody>
</table>