Dartmouth College
Computer Science 10, Winter 2012
Solution to Midterm Exam

Thursday, February 9, 2012
Professor Drysdale

Print your name: _____

- If you need more space to answer a question than we give you, you may use the backs of pages or you may use additional sheets of paper and attach them to the exam. Make sure that we know where to look for your answer!
- Read each question carefully and make sure that you answer everything asked for. Write legibly so that we can read your solutions. Please do not write anything in red.
- We suggest that for solutions that require you to write Java code, you include comments. They will help your grader understand what you intend, which can help you get partial credit.
- Hand in only what you want graded. We do not want your scrap paper unless it contains solutions you want us to grade.

Question	Value	Score
1	24	
2	16	
3	15	
4	16	
5	14	
6	15	
Total	100	

Question 1 24 points

Short answer

(a) (4 points)
In the k-means program we initialized the color list with k unique colors from the picture. After the first clustering there were no empty clusters. Yet later empty clusters sometimes arise. Explain how this can happen.

Solution: A cluster could have nothing very near its centroid. An example would be two tight sub-clusters of colors that got put into the same cluster. (Perhaps one sub-cluster contained an original color and the other sub-cluster was not near any of the original colors but was closer to the this clusters's original color than to the others.) In this case the centroid could be half way between the groups. But other new centroids are also computed, and it could be that each of the sub-clusters is closer to a another cluster's new centroid than to its own cluster's new centroid. In that case the new centroid could get no pixel colors assigned to it in the next reclustering and its cluster would then be empty.

(b) (4 points)
Describe an efficient way to implement a queue in an array.

Solution: Treat the array as a circular list. Keep variables `f` and `r` that keep track of the first item and one beyond the last item in the queue. Then `f` increases by on when enqueueing and `r` increases by one when dequeueing. However, when you get to the end wrap back to the beginning.

(c) (4 points)
Describe the difference between an ADT and a data structure. Give an example of each. How are they related?

Solution: An ADT is a collection of operations, but the exact way that the data is stored inside of it is hidden. Examples include `List` and `Map`. A data structure is a concrete way to store the data in the computer. Examples include `SentinelDLL` and arrays. Data structures are used to implement ADTs.

(d) (4 points)
Abstract classes and interfaces are similar in many ways. What things are allowed in an abstract class that are not allowed in an interface? What is the reason to have interfaces in Java, when everything possible in an interface is possible in an abstract class?

Solution: An abstract class can declare instance variables and can have constructors and non-abstract methods. You can only extend one abstract class, but can implement as many interfaces as you like.

(e) (4 points)
We have seen the Template design pattern. Explain this pattern and describe a class example where we used it.

Solution: The Template design pattern arises when several methods have basically the same overall structure, but differ in details. The overall structure is implemented at the top level (the template), but this method calls abstract methods (or trivial methods) to supply details. Then each subclass customizes the template by implementing the abstract methods.

An example is the binary operations implementing the Expression interface. All of them needed to have instance variables for a first and second operand. All of them needed to evaluate each operand and combine them using an operator. The difference between them was which operator to use. Therefore we had the class BinaryOp that implemented the general pattern, but the operation was performed by an abstract function `doOperation`. A specific binary operator (e.g. Sum) extend BinaryOp and provided an implementation of `doOperation` that performed the operation (e.g. adding) on its arguments. (Another example is the Shape abstract class in PS-3.)

(f) (4 points)
The Stack class in the Java library extends the Vector class. Vector is an earlier version of ArrayList with similar operations (e.g. get, add, contains, set, remove). The Stack class then implements the operations empty, peek, pop, and push. What principles of ADTs and Object-Oriented design does this violate? Why is this bad?

Solution: An ADT is defined by its operations. You should only be able to perform empty, peek, pop, and push on a stack. But if Stack extends Vector, then it is possible to call add, set, remove, contains, and all of the other methods in Vector on the stack. Thus you could change what is in the middle of the stack instead of only what is on the top, and this violates the contract of the stack ADT. The problem is that inheritance should only be used for an is-a relationship, and a Stack is not a Vector.

Question 2 16 points

Below is some code for a singly-linked (non-circular) list class that does not use a sentinel or dummy list head node. It does not have a tail reference or a current reference. The data stored in each node is an `int`. Note that in this representation, the last element in the list is indicated by having a next pointer of `null`. We will ask you to implement two methods for this class.

Here is the code:

```
public class SimpleSLL {
  private Element head;        // The first element of the list

  /**
   * A private class inner representing the elements in the list.
   */
  private class Element {
    public int data;
    public Element next;

    /**
     * Constructor for a linked list element, given an object.
     * @param obj the data stored in the element.
     */
    public Element(int theData) {
      data = theData;
      next = null;
    }
  }

 /**
   * Constructor to create an empty singly linked list.
   */
  public SLL() {
    head = null;
  }

 /**
   * Is this list empty?
   */
  public boolean isEmpty() {
    return (head == null);
  }

  // Other methods not shown
}
```

(a) (8 points)
Write a recursive method `sum` that, given a reference to an `Element elt`, will return the sum of data in the sublist whose head is referenced by `elt`. (Note that `elt` may refer to an `Element` in the middle of a list. In this case `sum` should return the sum of the data in the sub-list that starts at that `Element` and continues to the end of the list.) We define the sum of an empty list to be 0. Given this helper method we can write the sum the entire list by calling `sum(head)`. (Partial credit for a non-recursive solution.)

```
public int sum(Element elt) {
```

Solution:

```
  if(elt == null)
    return 0;
  else
    return elt.data + sum(elt.next);
}
```

(b) (8 points)
Write a method append that, given an `int value`, will append an `Element` containing `value` to the end of this list. Note that because you have no tail reference you must find the correct place to append.

```
public void append(int value) {
```

Solution:

```
  if(isEmpty())
    head = new Element(value);
  else {
    for(Element current = head;
        current.next != null;
        current=current.next)
      ;
    current.next = new Element(value);
  }
}
```

Question 3 15 points

Consider the following classes:

```
public class SuperClass {
  public void doIt() {
    doItMore();
    System.out.println("Super's doIt");
  }

  public void doItMore() {
    System.out.println("Super's doItMore");
  }

  public void doItSuper() {
    System.out.println("doItSuper");
  }
}

public class SubClass extends SuperClass {

  public void doIt() {
    super.doIt();
    System.out.println("Sub's doIt");
  }

  public void doItMore() {
    System.out.println("Sub's doItMore");
    super.doItMore();
  }

  public void doItSub() {
    System.out.println("doItSub");
  }
}
```

Assume that the following declarations are in a method in another class:

```
SuperClass superObj = new SuperClass();
SuperClass superObj2 = new SubClass();
SubClass subObj = new SubClass();
```

(a) (6 points)
For each of the following method calls, show what it would print or explain why the method call is not legal.

```
superObj.doItSuper();
    doItSuper

superObj.doItSub();
    Illegal, because doItSub not method of Super.

superObj2.doItSuper();
    doItSuper

superObj2.doItSub();
    Illegal, because doItSub is not a method of Super, and superObj2 is a variable
    of type Super (even though it holds a sub).

subObj.doItSuper();
    doItSuper

subObj.doItSub();
    doItSub
```

(b) (9 points)
Show the output from each of the following method calls:

```
superObj.doIt();

    Supers doItMore
    Supers doIt


superObj2.doItMore();

    Subs doItMore
    Supers doItMore


subObj.doIt();

    Subs doItMore
    Supers doItMore
    Supers doIt
    Subs doIt
```

Question 4 16 points

Write a method `findNode(E value)` which finds a `BinaryTree` whose element equals `value` in a heap-ordered binary tree. Assume that the type `E` implements `Comparable`, so that you can compare element values using `compareTo`. Write a recursive helper method.

A heap-ordered binary tree is a binary tree where the minimum value is at the root and a node's value is always smaller than its children's values. Note that this fact can under some circumstances let us avoid searching in a subtree. For full credit your method should take advantage of this and not search in subtrees that could not possibly contain a node whose element equals `value`. (Note that the values are stored in a binary tree. We can have the heap order property without storing the values in an array or an arraylist.)

```
/**
 * Searches for an element in this heap-ordered binary tree.
 * @param value the element value to search for.
 * @return a BinaryTree node whose element equals value, or null if no such
 *   node exists in this tree.
 */
public BinaryTree<E> findNode(E value) {
```

Solution

```
public BinaryTree<E> findNode(E value) {
    if(value.compareTo(getValue()) == 0)
      return this;
    else if (value.compareTo(getValue()) < 0)
      return null;  // If value < element at node it cannot be in the subtree.
    else {
      if(hasLeft()) {
        BinaryTree<E> leftReturn = getLeft().findNode(value);
        if(leftReturn != null)
          return leftReturn;
      }
      if(hasRight())
        return getRight().findNode(value);

      return null;  // Wasn't in either subtree.
    }
  }
```

Question 5 14 points

(a) (6 points)
The `TreeMap` `get` method takes $\Theta(\log n)$ time on average. You do a timing test and discover that performing 10,000 `get` operations on a map whose size is 1,000 takes 0.01 seconds. How long would you expect 20,000 `get` operations on a map whose size it 4,000,000 to take? (You may use the approximations $\log 1000 = 10$ and $\log 4,000,000 = 22$.)

Solution

We approximate the run time of a `get` operation to be $c \log n$. If $t(n)$ is the run time for the 10,000 `get` operations, this gives:

$t(n) = 10,000(c \log 1000) = 0.01$
$10,000(10)c = 0.01$
$c = 10^{-7}$

This means that the run time for 20,000 `get` operations on a map of size 4,000,000 is:
$20,000(10^{-7} \log 4,000,000 = 2(10^{-3}22) = 0.044$ seconds.

(b) (8 points)
Below is the RubberLines class, which we saw in class. The idea is that when the user presses the mouse one end of a segment is anchored there, and as long as the mouse is being dragged the other end follows the mouse. Add code to `init` to set up listeners and complete the `mousePressed` and `mouseDragged` methods to make this happen.

```
public class RubberLines extends JApplet implements MouseListener,
         MouseMotionListener {

  private Point point1 = null;
  private Point point2 = null;

  public void init() {
    // *** Fill in code to set up listeners ***

    addMouseListener(this);
    addMouseMotionListener(this);

    // Other code not shown  (and need not be written by you!)
  }

  public void mousePressed(MouseEvent event) {
  // *** Fill in the body of this method ***

    point1 = event.getPoint();

  }

  public void mouseDragged(MouseEvent event) {
  // *** Fill in the body of this method ***

    point2 = event.getPoint();
    repaint();

  }

  // Other code not shown

  private class Canvas extends JPanel {
    public void paintComponent(Graphics page) {
      super.paintComponent(page);
      if (point1 != null && point2 != null)
        page.drawLine(point1.x, point1.y, point2.x, point2.y);
    }
  }
}
```

Question 6 15 points

In SA-9 you wrote the class `StateAndCities`, which used a Map. In this map, the keys were names of states and the associated value for a key was a Set of names of cities that are within the state. Part of this class is shown below:

```
public class StatesAndCities
{
  private Map<String, Set<String>> stateCityMap;

  /**
   * Constructs empty map
   */
  public StatesAndCities()  {
    stateCityMap = new TreeMap<String, Set<String>>();
  }

  /**
   * Adds a state/city pair to the atlas.
   * @param state the state to add to
   * @param city the city to add
   */
  public void addPair(String state, String city)  {
    // Code not shown
  }

  /**
   * Is the city is associated with the state in the map
   * @param state the state to look for
   * @param city the city to look for
   * @return true if city is in state
   */
  public boolean isCityInState(String state, String city) {
     // Code not shown
  }
}
```

You are to add a method `getStatesContainingCity` to this class which, given the name of a city, returns a set of all states that contain a city with this name. (If none contain the city name, it returns the empty set.) Thus, if the current state of the map were:

```
MA: Boston  Concord
NH: Concord  Hanover
TN: Nashville
```

a call to `getStatesContainingCity("Concord")` should return the set {MA, NH}. You may use any of the methods given above if they are useful. The method is started on the next page.

```
  /**
   * Returns a Set containing the names of all states in the
   * stateCityMap whose associated Set contains the named city.
   * @param city the city that we seek
   * @return a set of states that contain city
   */
  public Set<String> getStatesContainingCity(String city)  {
```

Solution:

```
    Set<String> result = new TreeSet<String>();
    Set<String> stateSet = stateCityMap.keySet();

    for(String state : stateSet)
      if(isCityInState(state, city))
        result.add(state);

    return result;
}
```