**Your Name:**______________________

**I commit to uphold the ideals of honor and integrity by refusing to betray the trust bestowed upon me as a member of the Georgia Tech community.**

# CS 1316 Exam 2
# Summer 2009

| Section/Problem | Points Earned | Points Possible |
|---|---|---|
| 1. Terms & Concepts | | 64 |
| 2. Fill in the Blank | | 5 |
| 3. Trees | | 9 |
| 4. Code Comprehension | | 10 |
| 5. Turtle Graphics | | 8 |
| 6. Write Code: Copy A Queue | | 10 |
| 7. Write Code: AddLast | | 10 |
| 8. Write Code: Simple Plugin | | 15 |
| Total Points: | | 131 |

**Your Name:**______________________

## 1. Terms & Concepts (64 points)

For each of the terms below, write 1 or 2 sentences defining the term and proving you understand what your definition means. You may include an example if you think it will help your explanation. Be concise and precise.
General grading criteria: If the answer convinces the grader that the student knows what the term means, they receive 2 points. If the student uses some of the correct terminology but their answer is lacking enough details to convince the grader that they understand the term (but nothing about the answer is incorrect) they receive 1 point. If their answer includes incorrect elements mixed with correct elements, is clearly wrong, or missing, they receive zero points. Example answers that should receive 2 points are shown below. Typically in the examples below the first sentence is necessary and sufficient for full credit, the 2nd sentence adds details.

1. abstract (super) class – A class that can not be instantiated, but can be extended (subclassed). Can be used to define methods and behaviors that subclasses use.

2. abstract data type – A data type that has a description of how it works, but not how it is implemented. Abstract data types such as Stacks and Queues can be implemented in multiple different ways.

3. anonymous inner class – A class that has no name which can be quickly created and used to perform small jobs. An example is using an anonymous inner class to be the ActionListener for GUI components.

4. binary search tree – A Binary tree that is sorted so that items within it can be found quickly. Elements that are greater than a node are to the right of the node, elements that are smaller are to the left.

5. circular linked list – A linked list where the end points to the head. Can be used to store cyclical animations such as walking motion.

6. class – The blueprint or "framework" from which objects are instantiated. Classes define variables (fields) and behaviors ( methods) that every object will have, and sometimes also define static fields/methods that are stored in the class (shared by all objects).

7. dynamic data structure – A data structure that can grow and change. Examples include linked lists and trees.

8. final static variable – A constant variable that can't be changed.

9. graph (directed, undirected, acyclic) – A data structure made up of nodes and connections between nodes (lines). Connections can be unidirectional or bidirectional, and can sometimes have associated weights.

10. in-order traversal (of a tree) – Visiting all elements of a tree recursively: At each node, visit the left subtree, then visit the node itself, then visit the right subtree.

**Your Name:**____________________

11. inheritance – The process of inheriting behavior (methods) and data (fields/ variables) from a superclass to a child or subclass. Inheritance is the main way that code reuse is achieved in OOP.

12. interface – A list of method names (and signatures) that a class must implement to be said to "implement" the interface. Multiple interfaces can be implemented by the same class.

13. layout manager – In charge of controlling how GUI elements (widgets) are displayed/arranged/laid-out in a panel. Examples include Border, Flow, and Box layout.

14. leaf node – A node in a tree that has no children. (Located at the edges of the tree, hence the "leaf" name.)

15. LIFO – An acronym that stands for Last In First Out. A Stack is LIFO.

16. linked list – A data storage mechanism where a series of nodes each have a bit of data, and a pointer to the node that comes after them.

17. object – An object is a collection of fields (variables) and behaviors (methods) that can be "instantiated" based upon a "blueprint" or "plan" from a class.

18. private (keyword) – A keyword that indicates that a field or method can only be accessed by the current class/object.

19. queue – Abstract data type where the first item put into the queue will be the first that will be taken out. Useful for modeling real life queues/lines.

20. recursion – When a method that calls itself. (Should have a terminating condition and work towards that terminating condition to be "good" recursion.)

21. refactor(ing) code – Changing or moving code to remove duplication (typically by moving it up the inheritance hierarchy) without changing the overall program behavior, usually to improve readability or maintainability.

22. reference – A pointer to a specific object.

23. scene graph – A graph used to organize and lay out elements in a 2D or 3D animation. By manipulating a node in the scene graph, you can affect all of it's child nodes at the same time.

24. spanning tree – A graph without cycles that contains all of the nodes of a larger graph that has cycles. Useful to make sure you visit all nodes in a graph only once.

25. static field (variable) – A variable owned by the "class" instead of any particular (object) instance of the class. All instances of an object share one static class variable, although a static field can be accessed without creating even one instance of the class.

**Your Name:**______________________

26. static method – A method owned by the class. See static field definition. Can not access object variables.

27. superclass – The class that the current class inherited from (extended). Sometimes called the "Parent" class.

28. this (keyword) – A special keyword that always contains a refernece to the object that "holds" the currently active code.

29. traverse – The process of doing the same operation (set data, get data, print data, etc...) to every item in a data structure, typically a linked list, graph, or tree.

30. user interface events – Event that is dispatched when the user does something to a GUI, like clicking a button, or moving the mouse.

31. void (keyword) – A keyword that indicates that a method will not return anything.

## 2. Fill in the Blank ( 5 points)

In Java, logical **and** is written using the ___&&______ symbol, and logical **or** is written using the _____||_____ symbol.

Assume that the Student class is a subclass of the Person class, and the Person class is a subclass of the Human class. A variable that is defined to be of type Person can refer to (hold) an object of type _Person/Student___ or type __Student/Person_____ but a variable defined to be of type Student can only refer to an object of type _Student___________.

## 3. Trees ( 9 points)

a. If the in-order traversal of the binary tree T is: A D B G C F E, draw the tree:

```
        g
    d       f
a   b       c   e
```

b. If the pre-order traversal of the binary tree T is: A D B G C F E, draw the tree:

```
        a
    d           c
b       g   f       e
```

c. If the post-order traversal of the binary Tree is A D B G C F E, draw the tree:

```
            e
    b               f
a       d       g       c
```

**Your Name:**______________________

## 4. Code Comprehension (10 points):

Consider the method **`mystery`** below that manipulates linked lists of integers. What does this code do? (hint: draw a linked list of integers, then apply the code to the example list. IntNode is a linked list node similar to AgentNode or LLNode, which contains integers.)

If q references the list, what is returned by `mystery(q, null)`?

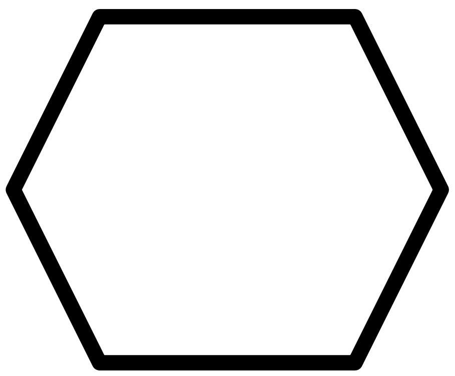
```
public IntNode mystery(IntNode aList, IntNode upTillNow ) {
   if (aList == null) {
       return upTillNow;
   } else {
       IntNode temp = aList.getNext();
       aList.setNext( upTillNow);
       return( mystery( temp, aList) );
   }
} This code returns the original list in reverse order.
  4 points if they get that it's recursive but not what it does.
```

## 5. Turtle Graphics ( 8 points)

**The following code creates a turtle, add the code to draw a hexagon like as the one shown to the right.** (You may draw your hexagon in any orientation, and with any length sides you want. You may start drawing the hexagon as soon as the turtle is created, you do not need to do any movement or rotation of the turtle to "get it to the hexagon" before you start drawing.)

```
Turtle t = new Turtle(new World());
for(int I = 0; I < 6; I++) {
   t.forward( <anyNumber>);
   t.turn(60);
 }
```

Grading:
4 points for turning at least 5 times
  -2 points if they didn't turn 60 degrees each time
4 points for moving exactly 6 times.
-2 points if they are off by one.



**Your Name:______________________**

## 6. Copy a queue (10 points):

Write code that accepts a queue as a parameter, creates a new empty queue, fills the new empty queue with the same contents as the original queue, and returns the new empty queue. Note: Your function must leave the original queue in the same state that it was in when you received it! You may assume the Queue supports the standard enqueue(item), dequeue(), peek(), size(), and isEmpty() methods.
We have given you the shell of the method, fill in the missing statements so that it works.

```
public static Queue<String> copyQ( Queue<String>  original) {
  // Create a new queue:
   Queue<String> newQ = new Queue<String>();
   int numElements = original.size();
   int doneSoFar = 0;
   while( doneSoFar < numElements) {
      // Add the missing statements here:

        String d = original.dequeue();
        newQ.enqueue(d);
        original.enqueue(d);
        doneSoFar++;    // OR, numElements--
```

Grading:

+3 points for getting items out of the original

+2 points for putting item into the newQ

+2 points for putting the item back into the original

+2 points for incrementing the counter.

```
    } // end while
    return(newQ);
} // end
```

**Your Name:**____________________

## 7. Write Code – Add Last (10 points)

Write a public method addLast( String data, StringNode first) that will create a new StringNode (containing the String "data") and add it to the end of the linked list (the "first" variable contains a reference to the first node in the linked list). You may assume that the first node will always exist (e.g. first will never be null). The class definition for StringNode is below:

```
public class StringNode {

    private myString;
    private StringNode next;

    public StringNode( String d) {
            myString = d;
            next = null;
    }

    public StringNode getNext() { return(next); }
    public void setNext( StringNode n) { next = n; }
    public void setData( String d) { myString = d; }
    public String getData() { return( myString );   }

 } // end class StringNode
```

```
public void AddLast( String data, StringNode first) {

     //Create the new node with data
     StringNode nn = new StringNode( data );

     // find the end of the linked list.
     StringNode current = first;   // optional! Could use first.
     while(current.getNext() != null) {
          current = current.getNext();
     }

     // Add to the end.
     current.setNext( nn );
}
```

Grading:

- 2 points for proper header.
- 2 point for creating the new node with the data.
- 4 points for correctly finding the last node of the list.
- 2 points for adding our new node to the end of the list.

**Your Name:______________________**

## 8. Create an Object: SimplePlugin (15 points)

Below is an interface for writing a plugin for an imaginary program. Your task is to write a class called SimplePlugin (the plugins' name is also "SimplePlugin") that implements PluginDesign but also contains two additional methods: one called "store" and one called "match":

- The "store" method accepts a generic object (of type "T") as it's only parameter and sets its value to a private generic class variable "myData" defined in the SimplePlugin class.
- The "match" method accepts a generic object (of type "T") as it's only parameter and compares it to the

private class variable "myData". It returns a true (Boolean) if they match (contain the same data, even if they are not the same object) and false (Boolean) otherwise.

You may assume that the generic object of type "T" has an equals() method. Be sure to consider the case when your SimplePlugin does not have an object stored.

```
public interface PluginDesign {
  //Return the name of your plugin
  public String getName();
  //Clears all class variables in the plugin
  public void resetVariables();
}
```

```
public class SimplePlugin<T> implements PluginDesign {
  //My field:
  private T myData;

  //My methods
  public void store( T data) {
    myData = data;
  }

  public Boolean match( T other) {
    if( myData == null) {
      return( false);
    } else {

      return(  myData.equals(other) );
    }
  }

  //PluginDesign Interface methods
  public String getName() { return("SimplePlugin"); }
  public void resetVariables() { myData = null;}


}
```

Grading:

- 2 points for header, one each for <T> and "implements PluginDesign".
- 1 point for having a "myData" field of type T
- 1 point if myData is private
- 3 points for having the store method that accepts T data and sets myData.
- 4 points for the match function: 1 each for the header, taking in a reference of type T, using myData.equals (NOT other.equals), and for checking if myData == null first.
- 4 points for the two interface methods. (2 each)