CIS 120 — Programming Languages and Techniques

Final Exam, May 3, 2011

Name: _____

Pennkey: _____

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

_____ _____

Signature Date

Scores:

1	
2	
3	
4	
5	
6	
7	
Total (120 max)	

1. **Dynamic Dispatch**

Consider the following **Java** class definitions:

```
class A {
    private int x;
    public A (int x0) { x = x0; }
    public int getX () {
        return x;
    }
}
class B extends A {
    private int x1;
    private int y;
    public B (int x0, int y0) {
        super (x0 + 1);
        x1 = x0;
        y = y0;
    }
    public int getX() {
        return x1;
    }
    public int getY() {
        return y;
    }
}
```

(a) (12 points) For each of the following variable declarations and initializations, list the static type of the variable and the dynamic class of the object referred to by that variable. If the initialization is invalid, write "INVALID" in both blanks. Assume that these statements occur in order, and that INVALID initializations do not prevent subsequent lines from referring to the variable.

	Static type	Dynamic class
`A a1 = new A(3);`	_____	_____
`B b1 = a1;`	_____	_____
`B b2 = new B(3,3);`	_____	_____
`A a2 = b2;`	_____	_____
`A a3 = new Object();`	_____	_____
`Object o1 = new B(2,3);`	_____	_____

(b) (10 points) Draw the state of the abstract stack machine just after the variable `z` has been pushed onto the stack. Be sure to include the class table. A sample ASM (from lecture 28) appears in the Appendix.

```
A b = new B(1,3);
int z = b.getX();
```

(c) (5 points) Briefly define the terms "static type" and "dynamic class" in Java. How are they related?

(d) (5 points) Briefly describe how the ASM model determines what code to put on the workspace at a method call.

2. **First-class functions**

(a) (5 points) Recall the OCaml function `map`:

```
let rec map (f : 'a -> 'b) (l : 'a list) : 'b list =
  begin match l with
  | [] -> []
  | (x :: xs) -> (f x) :: map f xs
  end
```

What is the value of the following OCaml expression?

```
map (fun (x : int) -> (x,x+1)) [1;3;5]
```

(b) (5 points) Now consider this OCaml function:

```
let rec foo (g : 'a -> 'a) (x : 'a) (n : int) : 'a =
  if n <= 0 then x else foo g (g x) (n-1)
```

What is the value of the following OCaml expression?

```
foo (fun (x : string list) -> "a" :: x) ["b"] 2
```

3. **Tree traversals**

(12 points) In this problem you will identify the behavior of a few simple tree functions in OCaml. Recall our OCaml definition of binary trees that store data in their internal nodes.

```
type 'a tree = Empty
             | Node of 'a tree * 'a * 'a tree
```

Here is an example of one such tree:

```
let example_tree : int tree =
   Node (Node (Node (Empty, 1, Empty),
               2,
               Node (Empty, 3, Empty)),
         4,
         Node (Node (Empty,5,Empty),
               6,
               Node (Empty,7,Empty)))
```

Now consider a few tree traversals:

(a)
```
let rec pre_t (t : 'a tree) : 'a list =
  begin match t with
  | Empty          -> []
  | Node (t1,a,t2) -> a :: (pre_t t1) @ (pre_t t2)
  end
in
  pre_t example_tree
```

Circle the result of this program.

```
[1; 2; 3; 4; 5; 6; 7]

[4; 2; 1; 3; 6; 5; 7]

[7; 6; 5; 4; 3; 2; 1]

[1; 3; 2; 5; 7; 6; 4]
```

(b)
```
let rec in_t (t : 'a tree) : 'a list =
  begin match t with
  | Empty          -> []
  | Node (t1,a,t2) -> (in_t t1) @ [a] @ (in_t t2)
  end
in
  in_t example_tree
```

Circle the result of this program.

```
[1; 2; 3; 4; 5; 6; 7]

[4; 2; 1; 3; 6; 5; 7]

[7; 6; 5; 4; 3; 2; 1]

[1; 3; 2; 5; 7; 6; 4]
```

(c)
```
let rec post_t (t : 'a tree) : 'a list =
  begin match t with
  | Empty          -> []
  | Node (t1,a,t2) -> (post_t t1) @ (post_t t2) @ [a]
  end
in
  post_t example_tree
```

Circle the result of this program.

```
[1; 2; 3; 4; 5; 6; 7]

[4; 2; 1; 3; 6; 5; 7]

[7; 6; 5; 4; 3; 2; 1]

[1; 3; 2; 5; 7; 6; 4]
```

(d)
```
let rec no_t (t : 'a tree) : 'a list =
  begin match t with
  | Empty          -> []
  | Node (t1,a,t2) -> (no_t t1) @ (no_t t2)
  end
in
  no_t example_tree
```

Write the result of this program.

4. **Program Design**

Use the four-step design methodology to implement a **Java** method, called `isAlmostCorrect`, which determines whether a `Word` is *almost* correctly spelled.

A word is almost correct if it is either already correct, or could be made to be correct by swapping two **adjacent** letters. For example, the word "cat" is almost correct. However, the word "cta" is also almost correct because it is one swap away from "cat". However, the word "tac" is not almost correct because there is no way to swap two adjacent letters to form a correctly spelled word.

The design process that we have been using this semester has four steps and you should use them to solve this design problem.

(a) (0 points) Step 1 is *understanding the problem.* You don't have to write anything for this part—your answers below will demonstrate whether or not you succeeded with Step 1.

(b) (4 points) Step 2 is *formalizing the interface.* The first part of formalizing the interface is to understand the interfaces of the classes that this method should interact with.

The argument to `isAlmostCorrect` should be an instance of the class `Word`, which represents sequences of letters. This class has the following interface:

```
public class Word {
    // Construct a word from the given string. If the argument string
    // contains nonletter characters (such as spaces or punctuation),
    // this constructor throws an InvalidArgumentException
    public Word (String w) { ... }

    // Return the number of letters in the word
    public int length() { ... }

    // Return a new word that exchanges the positions of the characters
    // at indices i and i+1.
    // For example, new Word("cta").swapAt(1) returns the word "cat"
    //
    // throws IllegalArgumentException when i is out of range
    //       (i.e. not 0 <= i < length()-1 )
    public Word swapAt(int i) { ... }

    // Compare two words ignoring their case
    public boolean equals(Object o) { ... }

    // Determine the ordering between words, ignoring their case
    public int compareTo(Object o) { ... }
}
```

The method `isAlmostCorrect` should be a member of the class `Dictionary`. This class contains a set of correct words, a constructor, and `isAlmostCorrect`. The declarations for these class members are at the top of the next page.

```
public class Dictionary {
    // set of correct words, stored in a binary search tree
    private TreeSet<Word> dict;

    // initializes dict with words from a file
    public Dictionary(String filename) { ... }

    // The method that you need to write
    public boolean isAlmostCorrect(Word word) { ... }
}
```

Although we have given you the method declaration for `isAlmostCorrect`, only **part** of the interface to this method is specified by this declaration. Below, complete the description of the interface to `isAlmostCorrect` by describing any exceptions that could be thrown and any shared state it must access. There are several correct answers here, but your answer to this problem must match your steps 3 and 4.

(c) (8 points) Step 3 is *writing test cases*. Create at least **four** new test cases with examples of the expected behavior, in addition to the one we have done for you.

Note that some test cases are better than others, and credit will be assigned accordingly: make sure your tests cover a sufficiently broad range of "interesting" inputs. Also, we have not specified what words are considered "correct". For the purposes of this problem, consider any common, normally uncapitalized English word to be correct. If it is unclear in your test cases whether a word is correct or not, please document your assumptions.

`assertTrue(isAlmostCorrect(new Word("cta")));`
Given from the problem description.

(d) (12 points) Step 4 is *implementing the program*. Implement the `isAlmostCorrect` method to complete the design. For reference, documentation for the Java Collections class `TreeSet` appears in the Appendix.

Linked data structures

The next few questions concern an implementation of mutable queues in Java. The classes `Queue`, `QueueNode`, and `EmptyQueueException` are shown in the Appendix. This implementation is a slight variant of the code that we discussed in class and lecture, so read through the classes now *before* continuing on to the problems below.

5. (10 points) Circle the result of each of the following code snippets.

(a)
```
Queue<String> q = new Queue<String>();
q.enq("a");
q.enq("b");
System.out.println(q.deq());
```

prints `a` prints `b` does nothing

throws `NullPointerException` throws `EmptyQueueException`

(b)
```
Queue<String> q1 = new Queue<String>();
Queue<String> q2 = new Queue<String>();
q1.enq("a");
q2.enq("b");
q1.deq();
System.out.println(q1.deq());
```

prints `a` prints `b` does nothing

throws `NullPointerException` throws `EmptyQueueException`

(c)
```
Queue<String> q1 = new Queue<String>();
Queue<String> q2 = q1;
q1.enq("a");
q2.enq("b");
q1.deq();
System.out.println(q1.deq());
```

prints `a` prints `b` does nothing

throws `NullPointerException` throws `EmptyQueueException`

(d)
```
Queue<Queue<String>> q1 = new Queue<Queue<String>>();
Queue<String> q2 = new Queue<String>();
q2.enq("a");
q1.enq(q2);
q2.enq("b");
System.out.println(q1.deq().deq());
```

prints `a` prints `b` does nothing

throws `NullPointerException` throws `EmptyQueueException`

(e)
```
Queue<Queue<String>> q1 = new Queue<Queue<String>>();
Queue<String> q2 =  new Queue<String>();
q1.enq(q2);
q1.enq(q2);
Queue<String> q3 = q1.deq();
q3.enq("a");
q2.enq("b");
System.out.println(q1.deq().deq());
```

prints `a` prints `b` does nothing

throws `NullPointerException` throws `EmptyQueueException`

6. The next two parts ask you to extend the Java `Queue` class shown in the Appendix with a new method, called `insert`.

The method `insert(n,v)` inserts an element `v` into the n^{th} position of the queue. To do so, it should create a new queue node containing `v` and add it into the queue so that it occurs at position `n`, where the head is at position 0. If `n` is a negative number, then the new element should be inserted at the beginning of the queue. If `n` is larger than the size of the queue, then the new element should be inserted at the end of the queue.

(a) (8 points) Briefly describe **four** test cases for the insert method. You don't have to give the complete code for these test cases, just describe what they should test in words.

i.

ii.

iii.

iv.

(b) (14 points) Now implement the insert method.

```
public void insert(int n, E v) {

}
```

7. **GUIs and Reactive Programming**

(10 points) This question concerns the Java Paint program from lecture and lab. For reference, an outline of this program appears in the Appendix. For each of the following statements, circle True or False.

(a) A good use of inner classes is when two classes need to share private state.

True False

(b) The class `Canvas` is an inner class of the class `Paint` and can access all private fields and methods of that class.

True False

(c) The fields `thinStroke` and `thickStroke` cannot be modified by any method, except for those in the class `Paint`.

True False

(d) The `mousePressed` method in the class `Mouse` is called by the Swing event loop whenever the user clicks a mouse button while the pointer is in the `Canvas` part of the screen.

True False

(e) It is the responsibility of the `mousePressed` method to draw a newly added shape on the canvas.

True False

(f) It is impossible to test Graphical User Interfaces.

True False

(g) The `paintComponent` method in the class `Canvas` only executes the very first time the canvas is displayed, at the start of the application.

True False

(h) The same object can be both a `MouseListener` and a `MouseMotionListener`.

True False

(i) In the `Paint` constructor, the purpose of the local variable `pane` is to help position the GUI elements on the screen by grouping together the two toolbars.

True False

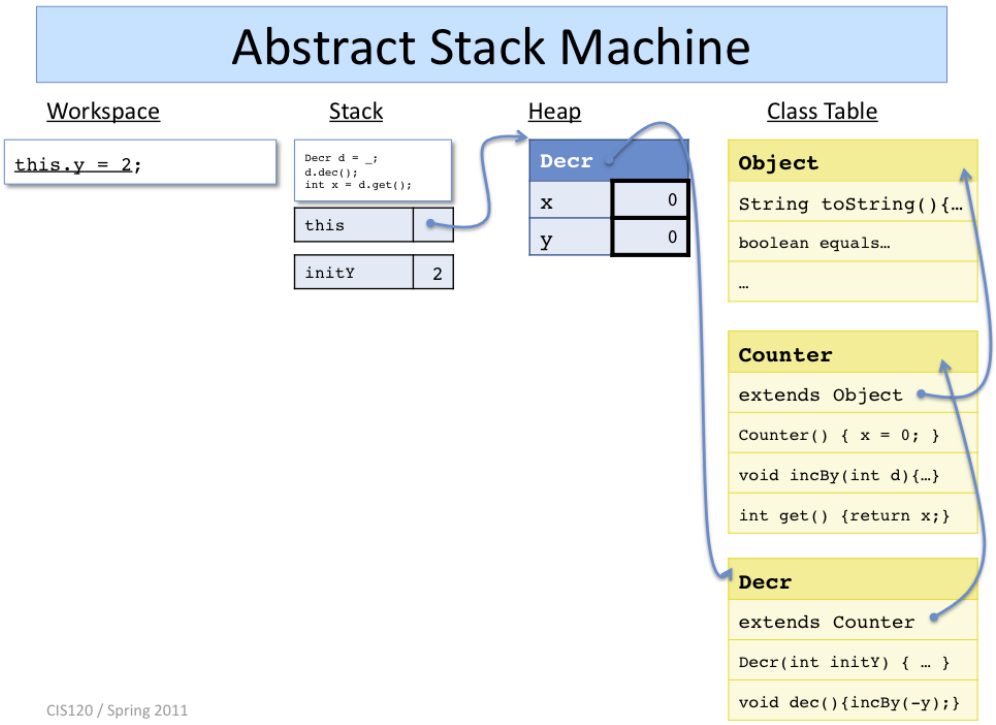
(j) A `JPanel` cannot be added to another `JPanel`.

True False

For Reference: Documentation for class `java.util.TreeSet<E>`

```
class java.util.TreeSet<E>

  boolean add(E e)
     Adds the specified element to this set if it is not already
     present. Returns true if the set did not already contain the
     specified element.

  boolean contains(Object o)
     Returns true if this set contains the specified
     element. More formally, returns true if and only if this set
     contains an element e such that (o==null ? e==null : o.equals(e)).

  boolean isEmpty()
     Returns true if this set contains no elements.

  int size()
     Returns the number of elements in this set.

  boolean remove(Object o)
     Removes the specified element from this set if it is present.
     Returns true if the set contained the specified element.
```

Sample Java ASM with Class Table (from Lecture 28)



For Reference: `Queue`, `EmptyQueueException` and `QueueNode`

The logic of these classes is the same as the code presented in class. The differences are (a) there is no division between the Queue interface and the QueueImpl implementation (b) deq throws an exception when called on an empty queue (c) fields in QueueNode are public, there are no getter or setter methods.

```
/** Nodes in the linked list */
class QueueNode<E> {
      public E v;
      public QueueNode<E> next;

      public QueueNode(E v0, QueueNode<E> next0) {
          v = v0;
          next = next0;
      }
  }

/** Thrown when deq invoked on an empty queue. */
class EmptyQueueException extends RuntimeException { }

class Queue<E> {

  /** A reference to the first node in the queue */
  private QueueNode<E> head;
  private QueueNode<E> tail;

  /** Create an empty queue */
  public Queue() {
    head = null;
    tail = null;
  }

  /** isEmpty() is true when the list has no elements */
  public boolean isEmpty() {
      return head == null;
  }

  public void enq(E x) {
      QueueNode<E> newnode = new QueueNode<E>(x, null);
      if (tail == null) {
         head = newnode;
         tail = newnode;
      } else {
         tail.next = newnode;
         tail = newnode;
      }
   }
```

```
  /** Retrieves and removes the first element of this queue. This
        method fails if the queue is empty. */
  public E deq() {
      if (head != null) {
         throw new EmptyQueueException();
      }
      E x = head.v;
      QueueNode<E> next = head.next;
      head = next;
      if (next == null) {
         tail = null;
      }
      return x;
   }
}
```

Outline of the Java `Paint` program.

Most of the method implementations are not shown, and are replaced with `...`.

```
public interface Shape {
  public void draw(Graphics2D gc);
}

public class PointShape implements Shape { ... }
public class LineShape implements Shape { ... }

public class Paint {
  /** Area of the screen used for drawing */
  private Canvas canvas;

  /** Current drawing color */
  private Color  color  = Color.BLACK;

  /** Strokes for drawing shapes with thin/thick lines */
  public final static Stroke thinStroke  = new BasicStroke(1);
  public final static Stroke thickStroke = new BasicStroke(3);

  /** Current drawing thickness */
  private Stroke stroke = thinStroke;

   /** Available modes for user input */
  public enum Mode {
    PointMode,
    LineStartMode,
    LineEndMode,
  }

  /** Current drawing mode */
  private Mode   mode   = null;

  /** Location of the mouse when it was last pressed. */
  private Point modePoint;

  /** The list of shapes that will be drawn on the canvas. */
  private LinkedList<Shape> actions = new LinkedList<Shape>();

  /** an optional shape for preview mode */
  private Shape preview;

  private class Canvas extends JPanel {
    public Canvas(){ ... {
      public void paintComponent(Graphics gc) { ... }
      public Dimension getPreferredSize() { ... }
  }
```

```
  /** Code to execute when the button is pressed while the mouse
   *  is in the canvas. */
   private class Mouse extends MouseAdapter {
        public void mousePressed(MouseEvent arg0){ [...] }
        public void mouseDragged(MouseEvent arg0) { [...] }
        public void mouseReleased(MouseEvent arg0) { [...] }
   }

  /** Create the toolbar containing radio buttons for the input
      modes as well as the thickness checkbox, Undo and Quit buttons */
  private JPanel createModeToolbar(final JFrame frame) { [...] }

  /** Create the color toolbar */
  private JPanel createColorToolbar() { [...] }

   /** Creates the main application window, assembles the user interface
      and displays it. */
  public Paint () {
    final JFrame frame = new JFrame();
    frame.setLayout(new BorderLayout());

    canvas = new Canvas();
    Mouse mouseListener = new Mouse();
    canvas.addMouseMotionListener(mouseListener); // dragged events
    canvas.addMouseListener(mouseListener);       // press/release events
    frame.add(canvas, BorderLayout.CENTER);

    JPanel pane = new JPanel();
    pane.setLayout(new GridLayout(2,1));
    frame.add(pane, BorderLayout.PAGE_END);
    pane.add(createModeToolbar(frame));
    pane.add(createColorToolbar());

    frame.pack();
    frame.setVisible(true);
  }

   /** Entry point for the Paint application. */
  public static void main(String[] args){
    SwingUtilities.invokeLater(new Runnable() {
      public void run() { new Paint(); }
    });
  }

}
```