Washington University **CSE131. Computer Science I**

Final Exam

Kenneth J. Goldman December 18, 2007

Name: _____

Student ID Number: _____

Signature: _____

Directions:

- This exam is closed book and closed notes. No electronic devices are permitted.
- Please check that you have *pages 1 through 12*. Write your name on every page.
- Do your own work. No discussion or collaboration with other students is permitted.
- If a question seems ambiguous, please walk to the front of the room and ask me about it quietly, or write an explanation of how you interpreted the question.
- The exam is divided into 11 sections, corresponding to Modules 2 through 10, plus optional extensions for Modules 3 and 4.
- **Complete only those sections for which you earned less than 85% on the corresponding quiz/midterm questions. Leave the rest blank.**
- **Under no circumstances should you work on more than seven sections of this exam.** If you attempt more than seven, they may not be graded.
- You have two hours to complete the exam.
- When you finish: *If fewer than 10 minutes remain, please do not turn in your exam early, since getting up may disturb other students who are trying to finish.*

Module 2
Recursion

```
(line 1)   int foo(int j, int k) {
(line 2)      if (j == k)
(line 3)         return j;
(line 4)      else
(line 5)         return 1 + foo(j+1, k);
(line 6)   }
```

1. Using the **substitution model**, show the execution of **foo(2, 4)**. Circle the return value.

2. In the execution of **foo(2, 4),** draw the **execution stack** as it would look immediately **after** the *first time line 3 is executed*.

3. Write a recursive method with the following specification. You may use the **++** and **--** operators, but you may **use neither addition nor subtraction**. (Also, you may **not** use **+=** or **-=**)

PARAMETERS: two integer values, **a** and **b**.
RETURN VALUE: the **absolute value** of the difference **a - b**

Bonus (1 point): Is your method tail recursive? _____ Explain: _____

Module 3
Iteration

Suppose a robot navigates in an infinitely large empty room as follows:

1. It starts at location (0,0).
2. It is given an initial deltaX and deltaY (both integers), and moves by that amount.
3. It then *cuts the deltaX and deltaY in half using integer division, and swaps their values.*
4. It moves again by the new deltaX and deltaY.
5. It keeps repeating steps 3 and 4 until it would move a distance of zero, and then stops.

Suppose you have a Point class with a constructor that takes integers x and y as parameters.

Using iteration, write a method that takes deltaX and deltaY as parameters, and returns a Point object representing the final position of the robot. **Do not use recursion.**

Write an iterative method with the following specification. You may use the **++** and **--** operators, but you may **use neither addition nor subtraction**. (You may **not** use **+=** or **-=** either.)

PARAMETERS: two integer values, **a** and **b**.
RETURN VALUE: the **absolute value** of the difference **a - b**

Module 3 Optional Extension
Loop Invariants

Consider the following method. Assume j >= 0.

```
int foo(int j, int k) {
   int p = 0;
   int count = 0;
   while (count < j) {
      p += k;
      count++;
   }
   return p;
}
```

1. What does this method compute? (If in doubt, try some small cases.)

2. State a loop invariant that would be useful in proving your conjecture above.

3. Use your loop invariant and the termination condition to argue that your conjecture is correct.

Module 4
Encapsulation

Consider the following Java code:

```
public void testAccount() {
   Account alice, bob, charlie, diane, edwin;
   alice = new Account(75);
   bob = new Account(75);
   alice.deposit(25);
   System.out.println("Alice has " + alice);
   System.out.println("Bob has " + bob);
   charlie = alice;
   alice.transfer(50, charlie);
   System.out.println("Alice has " + alice);
   System.out.println("Charlie has " + charlie);
   charlie = new Account(alice.getBalance());
   bob.transfer(50, charlie);
   System.out.println("Alice: " + alice);
   System.out.println("Bob: " + bob);
   System.out.println("Charlie: " + charlie);
   Account fred = new Account(5);
   fred.deposit(-10);
   System.out.println("Fred has " + fred);
   System.out.println("OK?"+fred.withdraw(-20));
   System.out.println("Fred has " + fred);
   alice.transfer(alice.getBalance(), charlie);
   System.out.println("Alice has " + alice);
   System.out.println("Charlie has " + charlie);
}
```

```
public class Account {
   private int balance;
   public Account(int openingBalance) {
      balance = openingBalance;
   }
   public int getBalance() {
      return balance;
   }
   public void deposit(int dollars) {
      balance = balance + dollars;
   }
   public boolean withdraw(int dollars) {
      if (dollars <= balance) {
         balance = balance - dollars;
         return true;
      } else
         return false;
   }
   public boolean transfer(int dollars,
                           Account dest) {
      if (withdraw(dollars)) {
         dest.deposit(dollars);
         return true;
      } else
         return false;
   }
   public String toString() {
      return ("$" + balance + ".00");
   }
}
```

1. How many Account **reference variables** are declared within the method `testAccount`? _____

2. How many Account **objects** are created during execution of the method `testAccount`? _____

3. Hand simulate the execution of the testAccount() method. Keep track of which object each variable refers to, and keep track of the values in each object. **Draw the variables and objects in the space below. Show each variable as a labeled rectangle and each object as a circle. Draw arrows from variables to the objects they reference.** As you go, fill in the blanks to complete the output that would be printed.

```
Alice has _____
Bob has _____
Alice has _____
Charlie has _____
Alice: _____
Bob: _____
Charlie: _____
Fred has _____
OK? _____
Fred has _____
Alice has _____
Charlie has _____
```

Module 4 Optional Extension
Test-Driven Development

Complete the following table to demonstrate your understanding of the iterative nature of test-driven development. Developer code should be the simplest reasonable implementation that will pass the test case on its left. Do not write a conditional statement for each assert statement.

Test Code	Implementation

`@Test`
`public void test() {`
`  LF c = new LF(2);`
`}`	

`@Test`
`public void test() {`
`  LF c = new LF(2);`
`  assertEquals(2, c.find(4));`
`  assertEquals(4, c.find(16));`
`}`	

`@Test`
`public void test() {`
`  LF c = new LF(2);`
`  assertEquals(2, c.find(4));`
`  assertEquals(4, c.find(16));`
`  LF c = new LF(3);`
`  assertEquals(2, c.find(9));`
`  assertEquals(3, c.find(27));`
`}`	

Module 5
Modular Design

Suppose you are given classes `Widget`, `Machine`, and `Truck`. Widgets have a color and a weight (in pounds), and are sometimes defective. Machines can transform widgets. When a machine transforms a widget, the method returns the same (albeit transformed) widget back as the return value. Some machines can make a widget become defective, or fix a defective widget. Machines may break when transforming widgets. Trucks can be loaded with widgets, one at a time. Widgets are loaded and unloaded at the back of the truck, so the widget that gets loaded first will be the last widget to be unloaded. So, if you load widgets a, b, and c into a truck, then you'll unload them in the order c, b, and a. If a truck is full, you can't load more widgets into it. If a truck is empty, you can't unload from it.

Widget	**Machine**	**Truck**
• Widget() • boolean isDefective() • double getWeight()	• Machine() • Widget transform(Widget w) • boolean isBroken()	• Truck() • void load(Widget w) • Widget unload() • boolean isEmpty() • boolean isFull() • double getTotalWidgetWeight()

1. Suppose that trucks crossing a bridge can carry at most 10,000 pounds of widgets. Suppose truck **t1** arrives at the bridge. Write Java code that creates additional trucks, as needed, and moves widgets from **t1** into those trucks, so that all of the trucks (including **t1**) can safely cross the bridge. Try not to create too many trucks, but don't be concerned about balancing the load equally among them. Assume that a widget can't weigh more than 10,000 pounds. Don't worry about keeping track of all the new trucks.

2. In your machine repair shop, you have two special widgets, **w1** and **w2**. If you let a broken machine transform **w1** up to ten times, it might fix the broken machine. Otherwise, letting it transform **w2** once will definitely fix the machine. However, if a machine is fixable by **w1**, then putting **w2** into the machine would cause an explosion. Similarly, putting either widget into a working (non-broken) machine will cause an explosion. Write a method takes a machine **m** and widgets **w1** and **w2** as parameters. When the method finishes executing, **m** should not be broken. Prevent explosions.

```
void fixAsNeeded(Machine m, Widget w1, Widget w2) {




}
```

CSE131 Module 6
Using Data Structures

1. Recall that if **s** is a String, then **s.length()** returns the number of characters in **s**. In a concise sentence, state the purpose of the following method. What is returned from foo, given the list ("Go", "bears!")?

```
LinkedList<Integer>  foo(LinkedList<String> ls) {
        LinkedList<Integer> result = new LinkedList<Integer>();
        Iterator<String> it = ls.iterator();
        while (it.hasNext())
            result.addFirst(it.next().length());
        return result;
}
```

2. Write an iterative method **eval** with the following specification. (**Do not use recursion.)**
   Parameters: a LinkedList<Double> **ls** and a double **x.** (Assume that **ls** is not null.)
   Return: f(x), where f is the polynomial whose coefficients are the values in **ls.**
   Example: If ls=(2 4 1) and x=5, then **eval** would return $2 + 4x + 1x^2 = 2 + 4*5 + 1*25 = 47$.
   Use a for loop to iterate of the elements of the list, but do not access them by index.
   Hint: Use a loop variable to keep track of powers of x.

CSE131 Module 7
Arrays

1. Write a method named **histogram** with the following specification.
   Parameters: **limit,** a non-negative integer
   data, an array of integers containing values in the range 0 to **limit-1**
   Return value: a new array where each slot *i* contains the number of times the value *i* appears in **data**

   Example: If the input is 5 and [2 4 1 2 0 1 1 4], the return value would be [1 3 2 0 2]
   because 0 occurs once in data, 1 occurs three times, 2 occurs twice, 3 never occurs, and 4 occurs twice.

2. Write a method named **similarity** with the following specification.
   Parameters: two boolean arrays of the same length
   Return value: the number of times the arrays have the same value in corresponding positions

   Example: If the given arrays are [true true false true] and [false true false true],
   the return value would be 3 because the two arrays agree at three positions

CSE131 Module 8
List Structures

1. Draw an object and reference diagram showing the data structure that would result after executing all of the following statements.

ListItem a = new ListItem(1, null);

ListItem b = new ListItem(2, new ListItem(3, a);

System.out.println("a = " + a);

System.out.println("b = " + b);

ListItem c = new ListItem(4, a.next);

b.next.next = c;

a = c;

System.out.println("a = " + a);

System.out.println("b = " + b);

System.out.println("c = " + c);

a ☐

b ☐

c ☐

```
public class ListItem {
 public int number;
 public ListItem next;

 public ListItem(int number, ListItem next){
     this.number = number;
     this.next = next;
 }

 public String toString() {
     if (this.next == null)
         return number;
     else
         return number + " " + next;
 }
}
```

2. What output would be printed by executing the code fragment from question 1?

3. Assuming that memory is allocated starting at memory address 200, complete the table at the right to show how the structure created by the code in question 1 would be represented in memory. **Put an X by the addresses of any memory cells that contain garbage objects.**

a ☐

b ☐

c ☐

200	
201	
202	
203	
204	
205	
206	
207	

CSE131 Module 9
Multiple Representations

1. Write an **append** method with the following specification.
   Parameters: **ls1**, the first ListItem of a list
   ls2, the first ListItem of another list
   Effect: put the list **ls2** at the end of list **ls1**
   Do not create any new list items. Assume ls1 is not null.

   Example: Given (9 8 7 6) and (5 4 3), the method would result in the combined list (9 8 7 6 5 4 3)

```
public class ListItem {
 public int number;
 public ListItem next;

 public ListItem(int number, ListItem next){
     this.number = number;
     this.next = next;
 }
}
```

2. Complete the following table. Show your work below.

The number…	In base…	Equals the number…	In base…
132	4		10
132	10		2
100101	2		8
61	8	100	

```
abstract class FactoryComponent {
  boolean on = false;
  FactoryComponent() {setOn(true);}
  boolean isOn() { return on; }
  void setOn(boolean b) { on = b; }
  public String toString() {
    return "FC (" + isOn() + ")";
  }
}
class Sensor extends FactoryComponent {
  double getReading() {return 0;}
  public String toString() {
    return super.toString() + " at " +
           getReading();
  }
}
class Actuator extends FactoryComponent{
  double sp;
  Actuator(double set) {
    setSetPoint(set);
  }
  public double getSetPoint() {
    return sp;
  }
  public void setSetPoint(double sp) {
    this.sp = sp;
  }
  public String toString() {
    return "Actuator at " +
           getSetPoint();
  }
}
class Thermometer extends Sensor {
  void setOn(boolean on) { }
  double getReading() { return 98.6; }
}
class Valve extends Actuator {
  Valve() { super(50); }
}
```

1. In the box, draw a class hierarchy for the above classes.

2. In the following code, put an "X" to the left of each line that would result in a **compilation error**, and put two stars ("**") to the left of each line that would compile but would result in a **run-time error**. In considering each line, assume that all the correct lines above it have executed.

```
FactoryComponent f1, f2, f3, f4;
f1 = new FactoryComponent();
f2 = new Sensor();
f3 = new Actuator(100);
f4 = new Thermometer();
Thermometer t;
t = f2;
t = (Sensor) f2;
t = (Thermometer) f2;
t = (Sensor) f4;
t = (Thermometer) f4;
Valve v = new Valve();
Actuator a = v;
v = a;
t = a;
t = (Thermometer) a;
f1 = t;
double d1, d2, d3;
d1 = f2.getReading();
d2 = ((Sensor) f3).getReading();
d3 = ((Sensor) f4).getReading();
```

3. What output would be printed by the following statements?

```
FactoryComponent f = new Sensor();
System.out.println(f.toString());
```

```
System.out.println(new Actuator(100));
```

```
System.out.println(new Thermometer());
```

```
Actuator act = new Valve();
System.out.println(act.toString());
```