CSU213 Exam 1 – Fall 2007

Name: ____________________

Student Id (last 4 digits): ____________________

• Write down the answers in the space provided.
• You may use all forms that you know from *ProfessorJ (Beginner)*, or *ProfessorJ (Intermediate*, where indicated. If you need a method and you don't know whether it is provided, define it.
• Remember that the phrase "develop a class" or "develop a method" means more than just providing a definition. It means to design them according to the design recipe. You are *not* required to provide a method template unless the problem specifically asks for one. However, be prepared to struggle if you choose to skip the template step.
• We will not answer *any* questions during the exam.

Good luck.

Problem	Points	/
1		/20
2		/20
3		/14
Total		/54

Problem 1

20 Points

A kindergarden teacher has a bag of beads for children to play with. The beads come in three colors, red, green, and blue, and in three different sizes: size 1, 2, and 3. We want to know more about the beads in the bag.

The following is the data definition for a bag of beads:

```
// to represent a bead
class Bead {
  String color;
  int size;

  Bead(String color, int size) {
    this.color = color;
    this.size = size;
  }
}

// to represent a bag of beads
interface Beads {
}

// to represent an empty bag of beads
class NoBeads implements Beads {

  NoBeads() {
  }
}

// to represent a nonempty bag of beads
class ManyBeads implements Beads {
  Bead one;
  Beads more;

  ManyBeads(Bead one, Beads more) {
    this.one = one;
    this.more = more;
  }
}
```

A. Make five examples of a bag of beads that you can use in the later parts of this problem.

Solution

[POINTS 3: must include empty bag, bag with one item, bag with multiple items. Note: These examples are used in subsequent tests.]

```
  Bead bs = new Bead("blue", 1);
  Bead bl = new Bead("blue", 3);
  Bead rm = new Bead("red", 2);
  Bead rs = new Bead("red", 1);
  Bead gm = new Bead("green", 2);
  Bead gl = new Bead("green", 3);


Beads none = new NoBeads();
Beads oneBlue = new ManyBeads(this.bs, this.none);
Beads oneRed = new ManyBeads(this.rm, this.none);
Beads many = new ManyBeads(this.bl,
               new ManyBeads(this.rm,
                 new ManyBeads(this.bs,
                   new ManyBeads(this.gl, this.none))));
```

B. Show the purpose, header, and a complete template for the method `blueBeads` that selects a bag of all blue beads from some bag of beads.

Solution

[POINTS 4: 1 for purpose and header, 1 for examples, 2 for the template]

```
// in the interface Beads:
// -- no template is possible - method has no body
// produce a bag of all blue beads in this bag of beads
Beads blueBeads();

// in the class NoBeads:
// -- no template is possible - the class has no fields
// produce a bag of all blue beads in this bag of beads
Beads blueBeads(){ return this;}


// in the class ManyBeads:
// produce a bag of all blue beads in this bag of beads
Beads blueBeads()
  ... this.one ...                -- Bead
  ... this.one.color ...          -- String
  ... this.one.size ...           -- String
  ... this.rest ...               -- Beads
  ... this.rest.blueBeads() ...   -- Beads
```

C. Complete the design of the method `blueBeads` that produces a bag of all blue beads from some bag of beads.

Solution

[POINTS 4: 1 for the body in the NoBeads class, 1 point for the body for the Beads class, 2 points for examples.]

```
// in the interface Beads:
// produce a bag of all blue beads in this bag of beads
Beads blueBeads();

// in the class NoBeads:
// produce a bag of all blue beads in this bag of beads
Beads blueBeads(){ return this; }


// in the class ManyBeads:
// produce a bag of all blue beads in this bag of beads
Beads blueBeads(){
  if (this.one.color.equals("blue")){
    return new ManyBeads(this.one, this.more.blueBeads());
  }
  else{
    return this.more.blueBeads();
  }
}

// int the class Examples:
// test the method blueBeads
boolean testBlueBeads =
         (check this.none.blueBeads() expect this.none) &&
         (check this.oneRed.blueBeads() expect this.none) &&
         (check this.oneBlue.blueBeads() expect this.oneBlue) &&
         (check this.many.blueBeads() expect this.allBlue);
```

D. Design the method `sortBeads` that produces a bag of beads sorted by their weight from the smallest to the largest.

Solution

[POINTS 9: 1 for purpose and header for both sort and insert in the interface; 1 point for method bodies for both methods in the NoBeads class; 1 point for the body of the sort method in the More class; 2 points for the body of the insert method in the More class; 2 points for examples/tests of the sort method; 2 points for examples/tests of the insert method – both sets of tests (must include empty, singleton, larger list, to cover both variants of the insert method)]

```
// in the interface Beads:
// produces a bag of beads sorted by their weight from small
// to medium to large from this bag of beads
Beads sortBeads();

// insert the given bead into this sorted bag of beads
Beads insert(Bead b);

// in the class NoBeads:
// produces a bag of beads sorted by their weight from small
// to medium to large from this bag of beads
Beads sortBeads(){ return this; }

// insert the given bead into this sorted bag of beads
Beads insert(Bead b){
  return new ManyBeads(b, this);
}


// in the class ManyBeads:
// produces a bag of beads sorted by their weight from small
// to medium to large from this bag of beads
Beads sortBeads(){
  return this.more.sortBeads().insert(this.one);
}
```

```
// insert the given bead into this sorted bag of beads
Beads insert(Bead b){
  if (b.size <= this.one.size){
    return new ManyBeads(b, this);}
  else {
    return new ManyBeads(this.one, this.more.insert(b)); }
}

// in the class Examples:
Beads sortedMany = new ManyBeads(this.bs,
                     new ManyBeads(this.rm,
                       new ManyBeads(this.bl,
                         new ManyBeads(this.gl, this.none))));

// test the method sortBeads
boolean testSort =
         (check this.none.sortBeads() expect this.none) &&
         (check this.many.sortBeads() expect this.sortedMany);

// test the method insert
boolean testInsert =
         (check this.none.insert(this.bs) expect this.oneBlue) &&
         (check this.sortedMany.insert(this.gm)
           expect new ManyBeads(this.bs,
                     new ManyBeads(this.gm,
                       new ManyBeads(this.rm,
                         new ManyBeads(this.bl,
                           new ManyBeads(this.gl, this.none))))));
```

Problem 2

20 Points

The network system consists of nodes that can be either routers or clients. Clients can send and receive messages through its input and output ports. Routers also have input and output ports, but additionally, each router has a list of its neighboring nodes it can communicate with. Every node has a unique name in the system. The input and output ports have names too, but we know nothing else about them.

For this exercise, the network is organized in such way that every router can be reached in a unique way from the start router, that means there are no cycles formed by the network connections.

A. Design the data representation for this information in the form of a class diagram. You do not need to include a purpose statement for each class or interface.

Solution

[POINTS 5: 1 for the Node interface or abstract class, 1 point for the Client class, 1 point for Router class, 1 point for the LoNode hierarchy, 1 point for correct arrows]

```
          +--------+
          | LoNode |<-----------------------+
          +--------+<-------------+        |
          +--------+              |        |
               |                  |        |
              / \                 |        |
              ---                 |        |
               |                  |        |
     -------------------          |        |
     |                 |          |        |
+----------+   +-------------+    |        |
| MTLoNode |   | ConsLoNode  |    |        |
+----------+   +-------------+    |        |
+----------+ +-| Node first  |    |        |
             | | LoNode rest |----+        |
             | +-------------+             |
             v                             |
           +------+                        |
           | Node |                        |
           +------+                        |
           +------+                        |
              |                            |
             / \                           |
             ---                           |
              |                            |
        ------------------------           |
        |                     |            |
+----------------+   +------------------+  |
| Client         |   | Router           |  |
+----------------+   +------------------+  |
| String name    |   | String name      |  |
| String inport  |   | String inport    |  |
| String outport |   | String outport   |  |
+----------------+   | LoNode neighbors |--+
                     +------------------+
```

B. Write down data examples, at least one for each class, representing at least four clients and three routers.

Solution

[POINTS 4: 1 point for examples for the Client class, 2 points for examples of routers, 1 point for examples of nonempty list of nodes.]

```
Node c1 = new Client("N1", "in", "out");
Node c2 = new Client("N2", "in", "out");
Node c3 = new Client("N3", "in", "out");
Node c4 = new Client("N4", "in", "out");
Node c5 = new Client("N5", "in", "out");

LoNode mtlist = new MTLoNode();
LoNode nodelist1 = new ConsLoNode(this.c1,
                     new ConsLoNode(this.c2, this.mtlist));

Node r1 = new Router("R1", "in", "out", this.nodelist1);

LoNode nodelist2 = new ConsLoNode(this.c3,
                     new ConsLoNode(this.c4,
                       new MTLoNode()));

Node r2 = new Router("R2", "in", "out", this.nodelist2);

Node r3 = new Router("R3", "in", "out",
                     new ConsLoNode(this.r1,
                       new ConsLoNode(this.r2,
                         new ConsLoNode(this.c5,
                           new MTLoNode()))));
```

C. Design the method `canReach` that determines whether some node (we know its name) can be reached from a router.

You are required to show templates for all classes that will be affected, including any methods you may need to put on your wish list.

Use separate pages for the templates, examples, and method definitions. Indicate clearly the class to which the templates belong, the class or interface for each method definition, and the class-method pair for each set of examples.

- Templates:

  ____________**Solution**____________

  [POINTS 3: 1 point for the Client class, 1 point for Router class, 1 point for the ConsLoNode, must include correct recursive calls]

```
/*---- in the Client class:   ----*/
     ... this.name ...                            -- String
     ... this.inport ...                          -- String
     ... this.outport ...                         -- String

     ... n.name ...                               -- String
     ... n.inport ...                             -- String
     ... n.outport ...                            -- String

/*---- in the Router class:   ----*/
     ... this.name ...                            -- String
     ... this.inport ...                          -- String
     ... this.outport ...                         -- String
     ... this.neighbors ...                       -- LoNode

     ... this.neighbors.canReach(String nodeName) ...     -- boolean

/*---- in the MTLoNode class:   ----*/
     ... nothing in the template ...

/*---- in the ConsLoNode class:   ----*/
     ... this.first ...                             -- Node
     ... this.rest ...                              -- LoNode

     ... this.first.canReach(String nodeName) ...   -- boolean
     ... this.rest.canReach(String nodeName) ...    -- boolean
```

- Examples:

  ____________**Solution** ____________

  [POINTS 4: 1 point for the Client class, 1 point for Router class, 1 point for the MTLoNode, 1 point for the ConsLoNode]

```
boolean testCanReachClient =
  (check this.c1.canReach("N2") expect false) &&
  (check this.c2.canReach("N2") expect true);

boolean testCanReachRouter =
  (check this.r1.canReach("N5") expect false) &&
  (check this.r1.canReach("N2") expect true) &&
  (check this.r2.canReach("N5") expect false) &&
  (check this.r3.canReach("N2") expect true) &&
  (check this.r3.canReach("N5") expect true);

boolean testCanReachMTLoNode =
  (check this.mtlist.canReach("N5") expect false);

boolean testCanReachConsLoNode =
  (check this.nodelist1.canReach("N5") expect false) &&
  (check this.nodelist1.canReach("N2") expect true) &&
  (check this.nodelist2.canReach("N5") expect false) &&
  (check this.nodelist2.canReach("N4") expect true);
```

- Methods:

Solution

[POINTS 4: 1 point for the Client class, 1 point for Router class, 1 point for the MTLoNode, 1 point for the ConsLoNode]

```
/*---- in the Client class:   ----*/
  // can the given node be reached from this node?
  boolean canReach(String nodeName){ return this.name.equals(nodeName); }

/*---- in the Router class:   ----*/
  // can the given node be reached from this node?
  boolean canReach(String nodeName){
    return this.neighbors.canReach(nodeName);
  }

/*---- in the MTLoNode class:   ----*/
  // can the given node be reached in this list?
  boolean canReach(String nodeName){ return false; }

/*---- in the ConsLoNode class:   ----*/
  // can the given node be reached from this node?
  boolean canReach(String nodeName){
    return this.first.canReach(nodeName) ||
           this.rest.canReach(nodeName);
  }
```

14 Points

Problem 3

Consider the following data definitions for a gift wrapping service:

```
// something about wrappings
interface Wrap {
}

// something wrapped more
class More implements Wrap {
  Wrap wrap;
  String color;

  More(Wrap wrap, String color) {
    this.wrap = wrap;
    this.color = color;
  }
}

// something inside
class Inside implements Wrap {
  String name;
  String contents;

  Inside(String name, String contents) {
    this.name = name;
    this.contents = contents;
  }
}
```

A. Draw a class diagram for the classes that represent the wrapping service data.

Solution

[POINTS 3: 1 for the Wrap interface, 1 for the two variants, 1 point for the arrows.]

```
               +------+
               | Wrap |<-------------------+
               +------+                    |
               +------+                    |
                   |                       |
                  / \                      |
                  ---                      |
                   |                       |
       -----------------------             |
       |                     |             |
+--------------+    +-----------------+    |
| More         |    | Inside          |    |
+--------------+    +-----------------+    |
| Wrap wrap    |-+  | String name     |    |
| String color | |  | String contents |    |
+--------------+ |  +-----------------+    |
                 |                         |
                 +-------------------------+
```

B. Make examples of data that represent the wrapping service data. Make sure your examples are sufficient to test properly the method `countBlue` defined in the last part of this problem.

Solution

[POINTS 4: 1 point for an example of Inside, 3 point for examples of More – should contain at least two wraps, should include wrap with no blue-s and a wrap with at least one blue wrap.]

```
Wrap gift = new Inside("Pete", "bike");
Wrap one = new More(this.gift, "blue");
Wrap two = new More(this.one, "red");
Wrap three = new More(this.one, "yellow");
Wrap four = new More(this.one, "green");
Wrap noblue = new More(
                new More(
                  new More(this.gift, "red"),
                "green"),
              "yellow");

Wrap twoblue = new More(
                 new More(
                   new More(this.gift, "red"),
                 "blue"),
               "blue");
```

C. Design the method `countBlue` that counts the blue colored `Wraps` in a `Wrap`

Solution

[POINTS 7: 1 point for purpose and header in the Wrap interface, 1 point for the body of the method in the class Inside, 2 points for the body of the method in the class More, 3 points for examples]

```
  // in the interface Wrap:
  // count the number of blue layers in this wrap
  int countBlue();

  // in the class Inside:
  // count the number of blue layers in this wrap
  int countBlue(){ return 0; }

  // in the class More:
  // count the number of blue layers in this wrap
  int countBlue(){
   if (this.color.equals("blue")){
     return 1 + this.wrap.countBlue();}
   else {
     return this.wrap.countBlue(); }
  }

  // in the class Examples:
 // tests for the method countBlue
 boolean testCountBlue =
   (check this.gift.countBlue() expect 0) &&
   (check this.one.countBlue() expect 1) &&
   (check this.noblue.countBlue() expect 0) &&
   (check this.twoblue.countBlue() expect 2);
```