CSU213 Exam 1 – Spring 2008

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		/18
2		/15
2 extra		/ 5
3		/12
Total		/45

18 Points

Problem 1

A children's computer game asks the child to collect magic flowers. The flower come in three colors, red, green, and blue. The magic power is hidden in the seed covered by a layer of petals. To gain the magic power the player must get through all layers of petals to pick the seed.

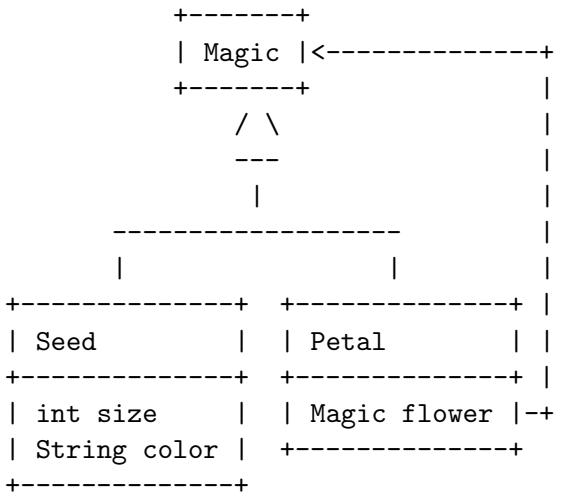
Here is a Scheme data definition for a magic flower:

```
;; A Magic is on of
;;  -- (make-seed Number String)
;;  -- (make-petal Magic)

(define-struct seed (size color))
(define-struct petal (flower))
```

A. Draw a class diagram for this data definition.

Solution [POINTS 2: must include three boxes with self-reference arrow.]



B. Complete the Java class definitions that represent these classes of data.

Solution [POINTS 2: must include an interface and two classes - with purpose statements.]

```
// to represent a magic flower
interface Magic{}

// to represent a seed of a magic flower
class Seed implements Magic{
  int size;
  String color;

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

// to represents a petal layer of the magic flower
class Petal implements Magic{
  Magic flower;

  Petal(Magic flower){
     this.flower = flower;
  }
}
```

C. Make examples of data for these classes.

Solution [POINTS 2: must include just seed, and seed covered with petals.]

```
class Examples{
  Examples(){}

  Magic redSeed = new Seed(3, "red");
  Magic blueSeed = new Seed(4, "blue");
  Magic greenSeed = new Seed(4, "green");

  Magic mid = new Petal(new Petal(this.blueSeed));
  Magic big = new Petal(this.redSeed);
  Magic small = new Petal(new Petal(new Petal(new Petal(this.blueSeed))));
  Magic mini = new Petal(new Petal(new Petal(this.greenSeed)));
}
```

D. Design the method `countPetals` that counts the number of petal layers in a magic flower.

_____**Solution** _____ [POINTS 3: purpose/header; bodies in both classes; examples]

```
// in the interface Magic:
  // count the number of petals covering the seed in this flower
  int countPetals();

// in the class Seed:
  // count the number of petals covering the seed in this flower
  int countPetals(){return 0;}

// in the classPetal:
  /* TEMPLATE:
    ... this.flower ...                -- Magic

    ... this.flower.countPetals() ...  -- int
  */

  // count the number of petals covering the seed in this flower
  int countPetals(){ return 1 + this.flower.countPetals(); }

// in the class Examples:
  // test the method countPetals in the classes that represent a magic flower
  boolean testCountPetals(){
    return (check this.redSeed.countPetals() expect 0) &&
           (check this.big.countPetals() expect 1) &&
           (check this.small.countPetals() expect 4);
  }
```

E. The power of the seed of a magic flower depends on the color of the flower. For red flowers, the power is three times its size, for blue flowers twice its size. For all other colors, the power of the seed is given by its size. The power of the flower is given by the power of its seed, but is diminished by one by every layer of petals that cover the seed.

Assume that red color is given as `"red"` and the blue color is given as `"blue"`.

Design the method `power` that compute the power of a flower. Make sure you follow the design recipe.

Include the template with your solution.

Solution [POINTS 7: 1 point purpose/header; 2 points for the template: own method invocation, self-reference method invocation - must include the type of the value. 2 points body; 2 points examples.]

```
// in the interface Magic:
  // find the seed of this flower
  Seed findSeed();

  // compute the magic power of this magic flower
  int power();

  // compute the magic power of this magic flower
  int power(){
    return this.flower.power() - 1;
  }

  //--------- alternative: the hard way ------------
  // in the class Seed:
  // find the seed of this flower
  Seed findSeed(){
    return this;
  }

  // compute the magic power of this magic flower
  int powerAlt(){
```

```
    if (this.color.equals("red"))
      return 3 * this.size;
    else if (this.color.equals("blue"))
      return 2 * this.size;
    else
      return this.size;
  }
  //--------- alternative: the end ----------------


// in the classPetal:
  /* TEMPLATE:
    ... this.flower ...                -- Magic

    ... this.countPetals() ...         -- int
    ... this.power() ...               -- int
    ... this.findSeed() ...            -- Seed

    ... this.flower.countPetals() ...  -- int
    ... this.flower.power() ...        -- int
    ... this.flower.findSeed() ...     -- Seed
  */

  // find the seed of this flower
  Seed findSeed(){
    return this.flower.findSeed();
  }

  // compute the magic power of this magic flower
  int power(){
    return this.findSeed().power() - this.countPetals();
  }

// in the class Examples:
  // test the method findSeed in the classes that represent a magic flower
  boolean testFindSeed(){
    return (check this.redSeed.findSeed() expect new Seed(3, "red")) &&
           (check this.mid.findSeed() expect new Seed(4, "blue")) &&
           (check this.mini.findSeed() expect new Seed(4, "green"));
  }
```

```
// test the method power in the classes that represent a magic flower
boolean testPower(){
  return (check this.redSeed.power() expect 9) &&
         (check this.mid.power() expect 6) &&
         (check this.big.power() expect 8) &&
         (check this.small.power() expect 4) &&
         (check this.mini.power() expect 1);
}
```

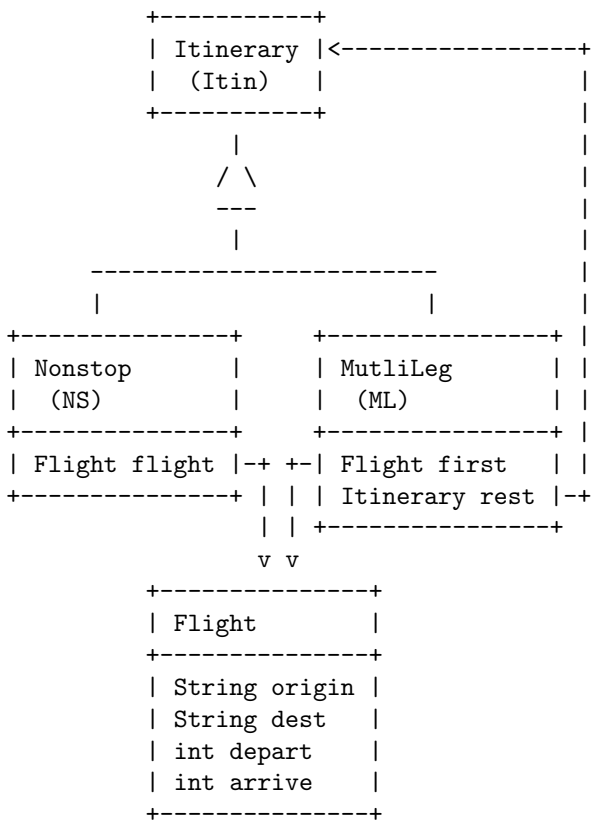
20 Points

Problem 2

The following class diagram allows us to represent information about flight itineraries. For example a flight from Boston to San Francisco may be a nonstop flight, or may involve several legs, changing planes in Philadelphia and in Houston. One leg of a flight that departs at 9 am and arrives at 2 pm may be given as:

```
Flight b2s = new Flight("BOS", "SFO",  9, 14);
```

Notice that we give only whole hour for either the departure time or the arrival time.



For the purpose of brevity you may use the names in parentheses for class names, instead of the full names given above.

A. Make an example of two nonstop itineraries and of the three-leg flight from Boston to San Francisco described earlier.

Solution [POINTS 2: 1 for the a nonstop flight, 1 point for three-leg flight.]

```
Flight bos_phi = new Flight("BOS", "PHI",  8, 10);
Flight phi_hst = new Flight("PHI", "HST", 11, 13);
Flight hst_sfo = new Flight("HST", "SFO", 14, 15);
Flight dfw_sfo = new Flight("DFW", "SFO", 14, 15);

Itinerary b2s = new Nonstop(new Flight("BOS", "SFO",  9, 14));
Itinerary p2h = new Nonstop(this.phi_hst);
Itinerary d2s = new Nonstop(new Flight("DFW", "SFO", 14, 15));
Itinerary b2s3 = new MultiLeg(this.bos_phi,
                   new MultiLeg(this.phi_hst,
                     new Nonstop(this.hst_sfo)));
Itinerary b2serr = new MultiLeg(this.bos_phi,
                     new MultiLeg(this.phi_hst,
                       new Nonstop(this.dfw_sfo)));
```

B. The traveler wants to know how much time will she spend at airports between landing of one flight and the takeoff of the connecting flight. Design the method `waitTime` that computes the total waiting time for an itinerary.

Solution [POINTS 7: 1 point purpose/header; 5 points body - including helper methods with tests; 2 points examples/tests.]

```
in the interface Itinerary:
 // compute the total waiting time between flights in this itinerary
 int waitTime();

 // compute the hours of waiting time for the first flight of this itinerary,
 // after the given time --- assume the given time is before the departure
 int hoursAfter(int time);

in the class Nonstop:
 // compute the total waiting time between flights in this itinerary
 int waitTime(){ return 0; }

 // compute the hours of waiting time for the first flight of this itinerary,
 // after the given time --- assume the given time is before the departure
 int hoursAfter(int time){
   return (this.flight.depart - time);
 }

in the class MultiLeg:
 // compute the total waiting time between flights in this itinerary
 int waitTime(){
   return this.rest.hoursAfter(this.first.arrive) +
          this.rest.waitTime();
 }

 // compute the hours of waiting time for the first flight of this itinerary,
 // after the given time --- assume the given time is before the departure
 int hoursAfter(int time){
   return (this.first.depart - time);
 }

in the class Examples:
```

```
// test the method waitTime in the classes that represent flight itinerary
boolean testWaitTime(){
  return (check b2s.waitTime() expect 0) &&
         (check b2s3.waitTime() expect 2);
}

// test the method hoursAfter in the classes that represent flight itinerary
boolean testHoursAfter(){
  return (check b2s.hoursAfter(9) expect 0) &&
         (check b2s3.hoursAfter(5) expect 3);
}
```

C. Design the method `isValid` that determines whether an itinerary is valid. For a multi-leg flight it means that next leg takes off from the same airport where the previous one landed, and leaves at least one hour after the arrival at this airport. A nonstop flight itinerary is always valid.

The first three steps of the design recipe are required for this problem. **The design of the body carries 5 points extra credit.**

Solution [POINTS 7: 1 point purpose/header; 5 points extra credit: body - including helper methods with tests; 2 points examples/tests.]

```
in the interface Itinerary:
 // is this a valid itinerary
 boolean isValid();

 // does this itinerary depart from the given airport?
 boolean departsFrom(String origin);

 // does this itinerary depart after the given time?
 boolean departsAfter(int time);

in the class Nonstop:
 // is this a valid itinerary
 boolean isValid(){
   return this.flight.isValid();
   // return true;
 }

 // does this itinerary depart from the given airport?
 boolean departsFrom(String origin){
   return this.flight.departsFrom(origin);
   // return this.flight.origin.equals(origin);
 }

 // does this itinerary depart after the given time?
 boolean departsAfter(int time){
   return this.flight.departsAfter(time);
   // return this.flight.depart > time;
 }
```

```
in the class MultiLeg:
 // is this a valid itinerary
 boolean isValid(){
   return this.rest.departsFrom(this.first.dest) &&
          this.rest.departsAfter(this.first.arrive) &&
          this.rest.isValid();
 }

 // does this itinerary depart from the given airport?
 boolean departsFrom(String origin){
   return this.first.origin.equals(origin);
 }

 // does this itinerary depart after the given time?
 boolean departsAfter(int time){
   return this.first.depart > time;
 }

in the class Examples:
 // test the method isValid in the classes that represent flight itinerary
 boolean testIsValid(){
   return (check b2s.isValid() expect true) &&
          (check b2serr.isValid() expect false) &&
          (check b2s3.isValid() expect true);
 }

 // test the method departsFrom in the classes that represent flight itinerary
 boolean testDepartsFrom(){
   return (check bos_phi.departsFrom("BOS") expect true) &&
          (check bos_phi.departsFrom("PHI") expect false) &&
          (check b2s.departsFrom("BOS") expect true) &&
          (check b2s.departsFrom("HST") expect false) &&
          (check b2s3.departsFrom("BOS") expect true) &&
          (check b2s3.departsFrom("HST") expect false);
 }

 // test the method departsAfter in the classes that represent flight itinerary
 boolean testDepartsAfter(){
```

```
  return (check bos_phi.departsAfter(7) expect true) &&
         (check bos_phi.departsAfter(8) expect false) &&
         (check b2s.departsAfter(7) expect true) &&
         (check b2s.departsAfter(9) expect false) &&
         (check b2s3.departsAfter(7) expect true) &&
         (check b2s3.departsAfter(9) expect false);
}
```

D. Show the templates for the classes `Nonstop` and `Multileg` after you have designed all of the methods.

Solution [POINTS 4: 1 point for the `Nonstop` class (must include data types); 3 points for the `Multileg` class (method invocations from the field `this.flight`, and self-referential, must include datatypes)]

```
/*---- in the Nonstop class:   ----*/
/* TEMPLATE:
  ... this.flight ...                      -- Flight

  ... this.flight.origin ...               -- String
  ... this.flight.dest ...                 -- String
  ... this.flight.depart ...               -- int
  ... this.flight.arrive ...               -- int

  ... this.hoursAfter(int time)...           -- int
  ... this.travelTime() ...                  -- int
  ... this.arrival() ...                     -- int
  ... this.flightTime() ...                  -- int
  ... this.isValid() ...                     -- boolean
  ... this.departsFrom(String origin) ...    -- boolean
  ... this.departsAfter(int time) ...        -- boolean
  ... this.destination() ...                 -- String
*/

/*---- in the MultiLeg class:   ----*/
/* TEMPLATE:
  ... this.first ...                     -- Flight
  ... this.rest ...                      -- Itinerary

  ... this.first.origin ...              -- String
  ... this.first.dest ...                -- String
  ... this.first.depart ...              -- int
  ... this.first.arrive ...              -- int

  ... this.hoursAfter(int time)...           -- int
  ... this.travelTime() ...                  -- int
  ... this.arrival() ...                     -- int
```

```
  ... this.flightTime() ...                  -- int
  ... this.isValid() ...                     -- boolean
  ... this.departsFrom(String origin) ...    -- boolean
  ... this.departsAfter(int time) ...        -- boolean
  ... this.destination() ...                 -- String

  ... this.first.hoursAfter(int time)...          -- int
  ... this.first.travelTime() ...                 -- int
  ... this.first.arrival() ...                    -- int
  ... this.first.flightTime() ...                 -- int
  ... this.first.isValid() ...                    -- boolean
  ... this.first.departsFrom(String origin) ...   -- boolean
  ... this.first.departsAfter(int time) ...       -- boolean
  ... this.first.destination() ...                -- String

  ... this.rest.hoursAfter(int time)...          -- int
  ... this.rest.travelTime() ...                 -- int
  ... this.rest.arrival() ...                    -- int
  ... this.rest.flightTime() ...                 -- int
  ... this.rest.isValid() ...                    -- boolean
  ... this.rest.departsFrom(String origin) ...   -- boolean
  ... this.rest.departsAfter(int time) ...       -- boolean
  ... this.rest.destination() ...                -- String
*/
```

This page is intentionally left blank - use it if you need extra space for any problem.

12 Points

Problem 3

You are trying to design a treasure hunting game that leads the player through a series of rooms. Each room has a door through which you enter and two other doors - one on the left and one on the right. Some doors are closed. Each open door leads to another room - again the same as the previous one. The size of the room does not matter, but each has a name and contains a treasure of certain weight.

To make our design easier to manage, every room may be entered only once - the player cannot get lost by returning to the same room over and over again.

The example below shows a sample maze:

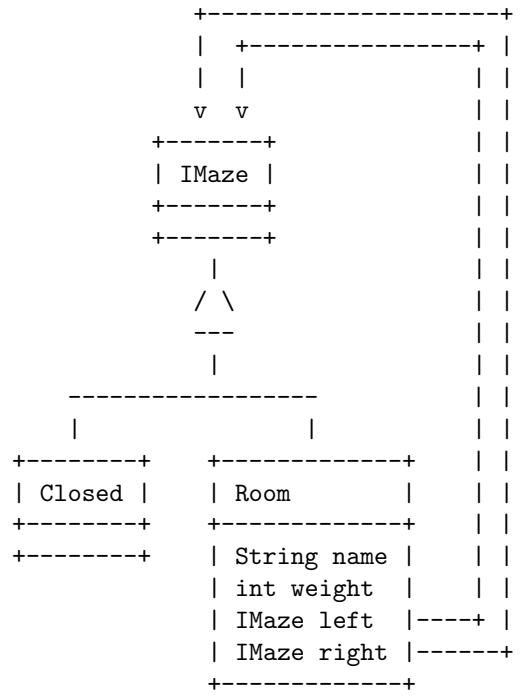
```
/*

 +-----R-----+-----------+-----L-----+
 |           |           |           |
 |   C/4           B/2          D/2  |
 |        IN L           R IN        |
 |                                   |
 |           |     IN    |           |
 +--- L -----+---- L ----+---- R ----+
   | IN   |  |           |  |  IN  |------L------+
   R      L  |           |  R      L IN     G/5  |
   |      |  |    A/3    |  |  F/2 |------R------+
   |  E/3 |  |           |  +------+
   +------+   IN         |
              |           |
              +-----R-----+
              |     IN    |-----L------+
              |                        |
              R           L IN    Y/10 |
              |   X/10                 |
              |           |-----R------+
              +-----------+
```

You can see that room A has 3 lbs of treasure and leads to the room B on the left. The right door from the room A is closed. Room B has 2 lbs of treasure and leads to the room C on the left and to the room D on the right...

A. Design the classes that represent such maze and represent your solution as a class diagram.

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



B. Show the complete Java class and interface definitions that correspond to your class diagram.

Solution [POINTS 5: 1 for the purpose statements; 1 for the IMaze interface; 1 for the closed door; 2 points for the room - with a constructor.]

```
// to represent a maze of rooms for a treasure hunt game
interface IMaze {
}

// to represent closed door in a maze
class Closed implements IMaze {
  Closed() {
  }
}

// to represent a room with two doors in a maze
class Room implements IMaze {
  String name;
  int weight;
  IMaze left;
  IMaze right;

  Room(String name, int weight, IMaze left, IMaze right) {
    this.name = name;
    this.weight = weight;
    this.left = left;
    this.right = right;
  }
}
```

C. Make examples of data that represent the maze shown above.

Solution [POINTS 4: 1 point for an example of a closed room, 3 points for examples of other rooms: subtract a point if rooms are missing, but those shown are correct (represent our maze), subtract two points if there are at least two examples of rooms that are correct, though they may not correspond to our maze.]

```
class Examples{
 Examples(){}

 IMaze closed = new Closed();

 IMaze roomG = new Room("G", 5, this.closed, this.closed);
 IMaze roomF = new Room("F", 2, this.roomG, this.closed);
 IMaze roomE = new Room("E", 3, this.closed, this.closed);
 IMaze roomD = new Room("D", 2, this.closed, this.roomF);
 IMaze roomC = new Room("C", 4, this.closed, this.roomE);
 IMaze roomB = new Room("B", 2, this.roomC, this.roomD);
 IMaze roomY = new Room("Y", 10, this.closed, this.closed);
 IMaze roomX = new Room("X", 10, this.roomY, this.closed);
 IMaze roomA = new Room("A", 3, this.roomB, this.roomX);
}
```