CS 2510 Exam 3 – Spring 2010

Name: _____

Student Id (last 4 digits): _____

• Write down the answers in the space provided.
• Your programs should work in standard Java 1.5. If you need a method and you don't know whether it is provided, define it.
• For tests you only need to provide the expression that computes the actual value, connecting it with an arrow to the expected value. For example `s.method() -> true` is sufficient.
• Remember that the phrase "design a class" or "design 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		/ 4
2		/ 6
3		/11
4		/14
5		/10
Total		/45

The next three problems will involve instances of the class `Bead` defined below:

```
/** To represent colored beads of different sizes */
class Bead{
  int size;
  String color;

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

In your work you may use the following examples of beads:

```
Bead b5r = new Bead(5, "r");
Bead b3r = new Bead(3, "r");
Bead b5g = new Bead(5, "g");
Bead b5b = new Bead(5, "b");
Bead b3b = new Bead(3, "b");
Bead b3g = new Bead(3, "g");
```

Of course, you may define additional examples as needed.

4 Points

Problem 1

Design a `Comparator` parametrized by the type `Bead` that compares two beads by the `String` that represents their color.

Remember, the interface `Comparator` is defined in the **Java Collections Framework** as

```
public interface Comparator<T>{
  pubic int compare(T t1, T t2);
}
```

Solution [POINTS 4: 1 points for the purpose statement; 1 point for the `compare` method, 1 point for defining the instance of `BComp` 1 point for any examples. Full points if the instance is defined as an anonymous class, and there are examples.]

```
/** To compare two beads by the String that defines their color */
class BComp implements Comparator<Bead>{
  public int compare(Bead b1, Bead b2){
    return b1.color.compareTo(b2.color);
  }
}

// in the class Examples:
  Comparator<Bead> bcomp = new BComp();

  void testComp(Tester t){
    t.checkExpect(this.leaf.comp.compare(this.b5r, this.b5b) > 0,
        true);
  }
```

6 Points

Problem 2

We are given an `ArrayList` of `Bead`s. In the class `Algorithms` design the method `isBad` that produces true if the `ArrayList` of `Bead`s contains a sequence of three beads with the colors `"r"`, `"g"`, `"b"`, immediately following each other (with no duplicates within). So, a sequence `"rggb"` or `"rgrb"` is OK.

Solution [POINTS 6: 1 point purpose statement, 3 points for correct solution, 2 points for tests that include true, false, starting sequence, ending sequence]

```
/**
 * Does the given list of beads contain
 * an rgb sequence of colors?
 * @param blist the given list of beads
 * @return true, if an rgb sequence is found
 */
boolean isBad(ArrayList<Bead> blist){
  for (int i = 0; i < blist.size() - 2; i++){
    if (blist.get(i).color.equals("r") &&
        blist.get(i+1).color.equals("g") &&
        blist.get(i+2).color.equals("b") )
      return true;
  }
  return false;
}

void testRGB(Tester t){
  ArrayList<Bead> blist1 = new ArrayList<Bead>();
  ArrayList<Bead> blist2 = new ArrayList<Bead>();
  ArrayList<Bead> blist3 = new ArrayList<Bead>();

  blist1.add(this.b3r);
  blist1.add(this.b3g);
  blist1.add(this.b3b);
  blist1.add(this.b3g);
  blist1.add(this.b3r);

  blist2.add(this.b3r);
  blist2.add(this.b3g);
  blist2.add(this.b3r);
  blist2.add(this.b3g);
```

```
  blist2.add(this.b3b);

  blist3.add(this.b3r);
  blist3.add(this.b3g);
  blist3.add(this.b3r);
  blist3.add(this.b3b);
  blist3.add(this.b3g);

  t.checkExpect(this.isBad(blist1), true);
  t.checkExpect(this.isBad(blist2), true);
  t.checkExpect(this.isBad(blist3), false);
}
```

11 Points

Problem 3

The class hierarchy below defines a binary search tree with a `Comparator`. One assumes that all examples of trees defined here will be *binary search trees*. However, the constructor for the class `Node` allows us to define binary trees that are not binary search trees.

Remember, that in a binary search tree all nodes in the left subtree are less than or equal to the root node, all nodes in the right subtree are greater than or equal to the root node, and every subtree is a binary search tree.

Design the method `isBST` for this class hierarchy that determines whether this tree is a binary search tree.

The *BST* class hierarchy:

```
// to represent a bunary search tree
abstract class ABST<T>{
  Comparator<T> comp;
}

// to represent a leaf in a binary search tree
class Leaf<T>extends ABST<T>{
  Leaf(Comparator<T> comp){
    this.comp = comp;
  }
}

// to represent a node in a binary search tree
class Node<T>extends ABST<T>{
  T data;
  ABST<T> left;
  ABST<T> right;

  Node(T data, ABST<T> left, ABST<T> right){
    if (left.comp.equals(right.comp)){
      this.comp = left.comp;
      this.data = data;
      this.left = left;
      this.right = right;
    }
    else
      throw new RuntimeException("Unmatched Comparators in a BST");
  }
}
```

You may use a binary tree of `Beads` for your tests.

Solution [POINTS 11: 2 points for purpose statements(includes 'this', 'given'), 6 points for the solution — needs at least one helper method, possibly two, 3 points for examples.]

```
// in the abstract class ABST<T>:

  // is this a binary search tree?
  abstract public boolean isBST();

  // do the data in all nodes in this tree
  // come before the given data
  abstract public boolean isBefore(T data);

  // do the data in all nodes in this tree
  // come after the given data
  abstract public boolean isAfter(T data);

// in the class Leaf<T>:

  // is this a binary search tree?
  public boolean isBST(){
    return true;
  }

  // do the data in all nodes in this tree
  // come before the given data
  public boolean isBefore(T data){
    return true;
  }

  // do the data in all nodes in this tree
  // come after the given data
  public boolean isAfter(T data){
    return true;
  }

// in the class Node:

  // is this a binary search tree?
```

```
  public boolean isBST(){
    return this.left.isBefore(this.data) &&
           this.right.isAfter(this.data) &&
           this.left.isBST() &&
           this.right.isBST();
  }

  // do the data in all nodes in this tree
  // come before the given data
  public boolean isBefore(T data){
    return (this.comp.compare(this.data, data) <= 0) &&
           this.right.isBefore(data);
  }

  // do the data in all nodes in this tree
  // come after the given data
  public boolean isAfter(T data){
    return (this.comp.compare(this.data, data) >= 0) &&
    this.left.isAfter(data);
  }

// in the class Examples:

  ABST<Bead> smalltree = new Node<Bead>(this.b5b, this.leaf, this.leaf);

  ABST<Bead> bstree =
    new Node<Bead>(this.b3g,
                   new Node<Bead>(this.b5g,
                                  new Node<Bead>(this.b5b,
                                                 this.leaf,
                                                 this.leaf),
                                  this.leaf),
                   new Node<Bead>(this.b3r,
                                  this.leaf,
                                  new Node<Bead>(this.b5r,
                                                 this.leaf,
                                                 this.leaf)));


  ABST<Bead> notbstree =
```

```
  new Node<Bead>(this.b3g,
                 new Node<Bead>(this.b5b,
                                new Node<Bead>(this.b5r,
                                               this.leaf,
                                               this.leaf),
                                this.leaf),
                 new Node<Bead>(this.b3g,
                                this.leaf,
                                new Node<Bead>(this.b5g,
                                               this.leaf,
                                               this.leaf)));

void testIsBST(Tester t){
  t.checkExpect(this.smalltree.isBST(), true);
  t.checkExpect(this.bstree.isBST(), true);
  t.checkExpect(this.notbstree.isBST(), false);
}

Bead b2a = new Bead(2, "a");
Bead b4t = new Bead(4, "t");

void testIsBeforAfter(Tester t){
  t.checkExpect(this.smalltree.isBefore(this.b4t), true);
  t.checkExpect(this.smalltree.isBefore(this.b2a), false);
  t.checkExpect(this.bstree.isBefore(this.b4t), true);
  t.checkExpect(this.bstree.isBefore(this.b2a), false);

  t.checkExpect(this.smalltree.isAfter(this.b4t), false);
  t.checkExpect(this.smalltree.isAfter(this.b2a), true);
  t.checkExpect(this.bstree.isAfter(this.b4t), false);
  t.checkExpect(this.bstree.isAfter(this.b2a), true);
}
```

Page for the solution to **Problem 3**

14 Points

Problem 4

The following two classes are a part of our game design:

```
// to represent a shot trying to hit a balloon
class Shot{
  int x;
  int y;

  Shot(int x, int y){
    this.x = x;
    this.y = y;
  }

  // did this shot hit the given balloon?
  boolean hitBalloon(Balloon b){
    return Math.abs(this.x - b.x) +
           Math.abs(this.y - b.y) <= 2;
  }
}

// to represent a balloon in a game
class Balloon{
  int x;
  int y;

  Balloon(int x, int y){
    this.x = x;
    this.y = y;
  }
}
```

A. **4 points**

Make examples of at least four `Balloon`s, four `Shot`s, and design a method that initializes an `ArrayList` of the four `Balloon`s and an `ArrayList` of the four `Shot`s to be used in tests.

Solution [POINTS 4: 1 for examples of shots, 1 for examples of Balloons, 2 for the init method]

```
Shot s24 = new Shot(2, 4);
Shot s35 = new Shot(3, 5);
Shot s28 = new Shot(2, 8);
Shot s14 = new Shot(1, 4);
Shot s83 = new Shot(8, 3);

Balloon b45 = new Balloon(4, 5);
Balloon b15 = new Balloon(1, 5);
Balloon b48 = new Balloon(4, 8);
Balloon b74 = new Balloon(7, 4);
Balloon b42 = new Balloon(4, 2);

ArrayList<Shot> slist = new ArrayList<Shot>();
ArrayList<Balloon> blist = new ArrayList<Balloon>();

void init(){
  this.slist.add(this.s24);
  this.slist.add(this.s35);
  this.slist.add(this.s28);
  this.slist.add(this.s14);
  this.slist.add(this.s83);

  this.blist.add(this.b45);
  this.blist.add(this.b15);
  this.blist.add(this.b48);
  this.blist.add(this.b74);
  this.blist.add(this.b42);
}
```

B. **10 points**

In the class `Algorithms` design the method `countHits` that counts the number of shots that have hit a balloon. If a balloon is hit by three shots, count three hits.

Solution [POINTS 10: 2 points for purpose statements, 3 points for countHits, 3 points for the helper method needed, subtract a point if loop within loop; 2 points for examples]

```
// in the class Algorithms:

  // count the number of times a shot in the given list
  // hits the given balloon - allowing for multiple hits
  // of one balloon
  int countHits(ArrayList<Shot> slist,
                ArrayList<Balloon> blist){
    int n = 0;
    for (Shot s: slist){
      n = n + countHitBalloons(s, blist);
    }
    return n;
  }

  // count the number of balloons hit by the given shot
  int countHitBalloons(Shot s,
                ArrayList<Balloon> blist){
    int n = 0;
    for (Balloon b: blist){
      if (s.hitBalloon(b))
        n = n + 1;
    }
    return n;
  }

// in the class Examples:

  Algorithms algo = new Algorithms();

  void testCountHits(Tester t){
    init();
    t.checkExpect(this.s14.hitBalloon(this.b15), true);
    t.checkExpect(this.s83.hitBalloon(this.b74), true);
    t.checkExpect(this.algo.countHits(this.slist, this.blist), 6);
  }
```

10 Points

Problem 5

Parametrize the solution to **Problem 4** as follows:

Parametrize the method `countHits` by the classes that represent two different `ArrayLists` of objects, and pass to the method `countHits` a function object that allows us to determine whether one object hit another one.

Solution [POINTS 10: 1 point purpose for the interface `Hit` (or the method within), 1 point for the interface definition; 1 point for an example of a class and object that implements `Hit`, 5 points for the method definitions; 2 points for examples for the method(s).]

```
// in the class Algorithms:

  // count the number of times a shot in the given list
  // hits the given balloon - allowing for multiple hits
  // of one balloon
  <R, T> int countHitsRT(ArrayList<R> rlist,
                ArrayList<T> tlist, Hit<T, R> hit){
    int n = 0;
    for (T t: tlist){
      n = n + countHitR(t, rlist, hit);
    }
    return n;
  }

  // count the number of balloons hit by the given shot
  <R, T> int countHitR(T t,
                ArrayList<R> rlist, Hit<T, R> hit){
    int n = 0;
    for (R r: rlist){
      if (hit.wasHit(t, r))
        n = n + 1;
    }
    return n;
  }

  // did object t hit the object r
  interface Hit<T, R>{
    public boolean wasHit(T t, R r);
  }

// in the class Examples:
```

```
Hit<Shot, Balloon> shb = new ShotHitBalloon();

void testCountHitsRT(Tester t){
  init();
  t.checkExpect(this.shb.wasHit(this.s14, this.b15),
      true);
  t.checkExpect(this.shb.wasHit(this.s83, this.b74),
      true);
  t.checkExpect(this.shb.wasHit(this.s83, this.b42),
      false);
  t.checkExpect(this.algo.countHitsRT(this.blist,
                                      this.slist,
                                      this.shb), 6);
}
```