CSU213 Exam 1 – Spring 2007

Name: _____

Student Id (last 4 digits): _____

Homework login name: _____

Instructor's Name + Time: _____

- 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		/10
3		/15
4		/15
Total		/60

20 Points

Problem 1

A bad apple spoils the whole bunch. We want to do something about bad apples in the bushel of apples. To do so, for each apple we keep track of whether it is good or bad, and how much does it weigh (in whole ounces). The bushel is represented as a list of apples.

The following is the data definition for a bushel of apples:

```
// to represent an apple
class Apple {
  boolean good;
  int weight;

  Apple(boolean good, int weight) {
    this.good = good;
    this.weight = weight;  }
}

// to represent a bushel of apples
interface LoA {
}

// to represent an empty bushel of apples
class MtLoA implements LoA {

  MtLoA() {  }
}

// to represent a nonempty bushel of apples
class ConsLoA implements LoA {
  Apple first;
  LoA rest;

  ConsLoA(Apple first, LoA rest) {
    this.first = first;
    this.rest = rest;  }
}
```

A. Make at least three examples of bushels of apples.

Solution [POINTS 3: must include empty list, lists of one item (good and bad), lists with multiple items, including all good and all bad. Note: These examples are used in subsequent tests. A point lost here may indicate insufficuent number of test cases for one of the subsequent problems.]

```
Apple good1 = new Apple(true, 8);
Apple good2 = new Apple(true, 5);
Apple bad1 = new Apple(false, 4);
Apple bad2 = new Apple(false, 6);

LoA mtbush = new MtLoA();
LoA bushOK = new ConsLoA(this.good1, this.mtbush);
LoA bushBad = new ConsLoA(this.good1, this.mtbush);
LoA bushOK2 = new ConsLoA(this.good1,
                  new ConsLoA(this.good2, this.mtbush));
LoA bushBad2 = new ConsLoA(this.bad1,
                  new ConsLoA(this.bad2, this.mtbush));
LoA bushMix = new ConsLoA(this.good1,
                  new ConsLoA(this.bad1,
                   new ConsLoA(this.good2,
                    new ConsLoA(this.bad2, this.mtbush))));
```

B. Show the purpose, header, and a complete template for the method `goodApples` that produces a list of all good apples in a bushel of apples.

Solution [POINTS 3: 1 for purpose and header, 1 for examples, 1 for the body]

```
// in the interface LoA:
// -- no template is possible - method has no body
// are there any bad apples in this bushel?
boolean anyBad();

// in the class MtLoA:
// -- no template is possible - the class has no fields
// are there any bad apples in this bushel?
boolean anyBad(){ ... }


// in the class ConsLoA:
// are there any bad apples in this bushel?
boolean anyBad(){
  ... this.first ...           -- Apple
  ... this.first.good ...      -- boolean
  ... this.first.weight ...    -- int
  ... this.rest ...            -- LoA
  ... this.rest.anyBad() ...   -- boolean
}
```

C. Complete the design of the method `goodApples` that produces a list of all good apples in a bushel.

Solution [POINTS 4: 1 for purpose, header, and the body in the empty case, 1 point for the body for the cons case, 2 points for examples: no points taken off here if both empty and nonempty cases are covered – see part A.]

```
// in the interface LoA:
// produce a list of all good apples in this bushel
LoA goodApples();

// in the class MtLoA:
// produce a list of all good apples in this bushel
LoA goodApples(){ return this; }


// in the class ConsLoA:
// produce a list of all good apples in this bushel
LoA goodApples(){
  if (this.first.good)
    return new ConsLoA(this.first, this.rest.goodApples());
  else
    return this.rest.goodApples(); }

// int the class Examples:
// test the method goodApples
boolean testGoodApples(){
  return
    (check this.mtbush.goodApples() expect this.mtbush) &&
    (check this.bushOK.goodApples() expect this.bushOK) &&
    (check this.bushBad.goodApples() expect this.mtbush) &&
    (check this.bushBad2.goodApples() expect this.mtbush) &&
    (check this.bushOK2.goodApples() expect this.bushOK2) &&
    (check this.bushMix.goodApples() expect this.bushOK2); }
```

D. Design the method `sortApples` that produces a list of all apples in a bushel sorted by their weight. **Do this problem last, even if you know exactly what to do.**

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 empty class; 1 point for the bosy of the sort method in the cons class; 2 points for the body of the insert method in the cons 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 LoA:
// produce a list of all apples in this bushel sorted by weight
// from this list of apples
LoA sortApples();

// insert the given apple into this sorted list of apples
LoA insert(Apple a);

// in the class MtLoA:
// produce a list of all apples in this bushel sorted by weight
// from this list of apples
LoA sortApples(){ return this; }

// insert the given apple into this sorted list of apples
LoA insert(Apple a){ return new ConsLoA(a, this); }


// in the class ConsLoA:
// produce a list of all apples in this bushel sorted by weight
// from this list of apples
LoApp sortApples(){
  return this.rest.sortApples().insert(this.first); }

// insert the given apple into this sorted list of apples
LoA insert(Apple a){
```

```
  if (a.weight < this.first.weight)
    return new ConsLoA(a, this);
  else
    return new ConsLoA(this.first, this.rest.insert(a)); }

// in the class Examples:
// test the method sortApples
boolean testSortApples(){
  return
    (check this.mtbush.sortApples() expect this.mtbush) &&
    (check this.bushOK.sortApples() expect this.bushOK) &&
    (check this.bushBad.sortApples() expect this.bushBad) &&
    (check this.bushOK2.sortApples() expect
       new ConsLoA(this.good2,
        new ConsLoA(this.good1, this.mtbush))) &&
    (check this.bushMix.sortApples() expect
       new ConsLoA(this.bad1,
        new ConsLoA(this.good2,
         new ConsLoA(this.bad2,
          new ConsLoA(this.good1, this.mtbush)))));
}

// test the method insert
boolean testInsert(){
  return
    (check this.mtbush.insert(this.good1) expect this.bushOK) &&
    (check this.bushOK.insert(this.good2) expect
       new ConsLoA(this.good2,
        new ConsLoA(this.good1, this.mtbush))) &&
    (check
       new ConsLoA(this.good2,
        new ConsLoA(this.good1, this.mtbush)).insert(this.bad1)
     expect
       new ConsLoA(this.bad1,
        new ConsLoA(this.good2,
         new ConsLoA(this.good1, this.mtbush)))) &&
    (check
       new ConsLoA(this.good2,
        new ConsLoA(this.good1, this.mtbush)).insert(this.bad2)
     expect
```

```
new ConsLoA(this.good2,
 new ConsLoA(this.bad2,
  new ConsLoA(this.good1, this.mtbush)))); }
```

10 Points

Problem 2

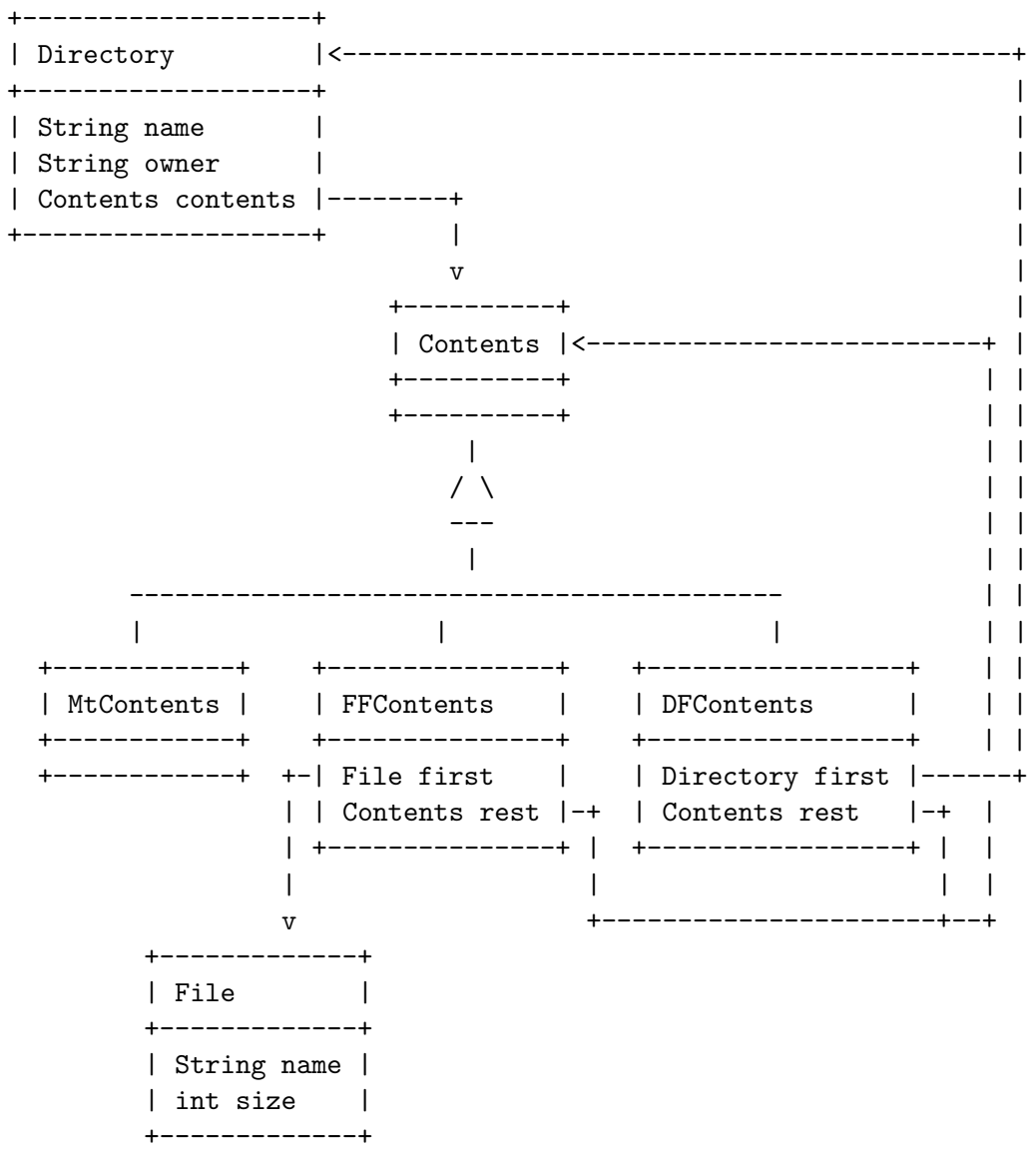
The system keeps track of all the files in the system and all the directories that contain both the files and also other directories. Your task is to design a data definition that would allow the system programmer to keep track of all this information.

Here are some details you need to know:

- For each file you record the file name and its size
- For each directory you record the directory name and the name of the owner and the directory contents
- The directory contents can be one of the following: it may be empty, it may contain a file followed by the remaining contents, or it may contain a directory, followed by the remaining contents

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 Directory class, 1 point for the Contents interface, 1 point for the three variants of the Contents, 1 point for the File class, 1 point for correct arrows]



B. Write down data examples, at least one for each class:

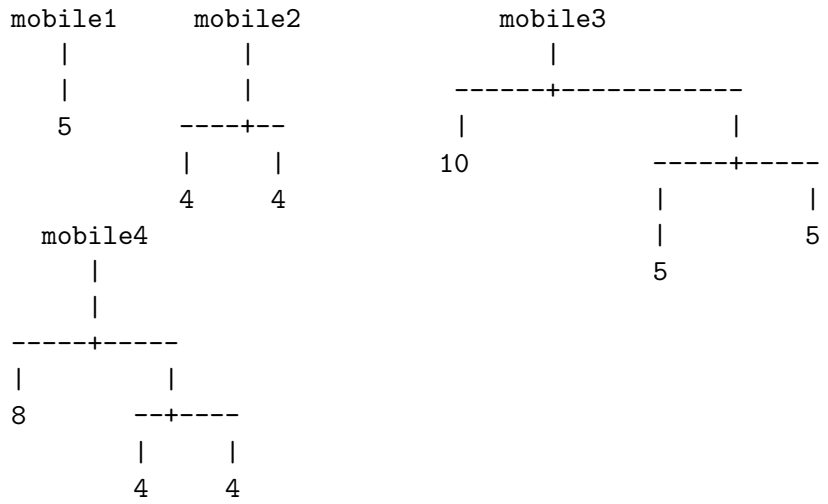
Solution [POINTS 5: 1 point for examples for the File class, 1 point for examples for the empty contents and contents with a file as the first element, 1 point for examples for the Directory class that consists of files only, 1 point for examples for the contents with a directory as the first element,1 point for examples for the Directory class that includes other directories.]

```
File file1 = new File("file1", 10);
File file2 = new File("file2", 20);
File file3 = new File("file3", 30);

Contents mtlist = new MtContents();
Contents f1list = new FFContents(this.file1, this.mtlist);
Contents f1f2list = new FFContents(this.file1,
                      new FFContents(this.file2, this.mtlist));

Directory mtdir = new Directory("MT", "Paul", this.mtlist);
Directory dirF1list =
    new Directory("F1-list", "Paul", this.f1list);
Directory dirF1F2list =
    new Directory("F1F2-list", "Paul", this.f1f2list);

Contents biglist1 = new DFContents(this.dirF1list, this.mtlist);
Contents biglist2 =
    new DFContents(this.dirF1F2list,
                   new FFContents(this.file2, this.mtlist));

Directory bigdir1 = new Directory("big1", "Tim", this.biglist1);
Directory bigdir2 = new Directory("big2", "Tim", this.biglist2);
```

15 Points

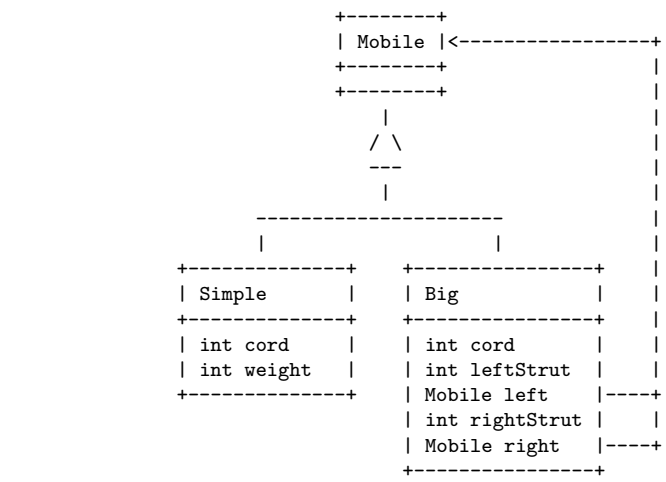
Problem 3

An artist is designing mobiles. The following pictures show examples of some mobiles:



The artist is not very happy with some of them - they do not look balanced. Your job is to design a program that will help the artist design balanced mobiles.

A. The class diagram represents all the information we need to know about mobiles. Study it carefully and write down the actual class and interface definitions in ProfessorJ beginner language.



Solution [POINTS 5: 1 for purpose statements for the interface and the two classes, 1 point for the interface definition, 1 point for the Simple class definition, 2 points for the Big class definition.]

```
// to represent a mobile
interface Mobile { }

// to represent a simple mobile
class Simple implements Mobile {
  int cord;
  int weight;

  Simple(int cord, int weight) {
    this.cord = cord;
    this.weight = weight; }
}

// to represent a nontrivial mobile
class Big implements Mobile {
  int cord;
```

```
  int leftStrut;
  Mobile left;
  int rightStrut;
  Mobile right;

  Big(int cord, int leftStrut, Mobile left,
                int rightStrut, Mobile right) {
    this.cord = cord;
    this.leftStrut = leftStrut;
    this.left = left;
    this.rightStrut = rightStrut;
    this.right = right; }
}
```

B. Design examples of data that represent mobiles. Include among the examples data that represents the mobiles shown in the introduction to this problem. *The lengths of the cords and struts should be in proportion to the given ones.*

Solution [POINTS 5: 1 for each of the four shown mobiles, 1 point for an extra example.]

```
// Examples from the exam:
Mobile mobile1 = new Simple(2, 5);
Mobile mobile2 = new Big(2,
                         4, new Simple(1, 4),
                         4, new Simple(1, 4));
Mobile mobile3 =
  new Big(1,
          5, new Simple(1, 10),
          10, new Big(1,
                      5, new Simple(2, 5),
                      5, new Simple(1, 5)));
Mobile mobile4 =
  new Big(1,
          5, new Simple(1, 8),
          5, new Big(1,
                     2, new Simple(1, 4),
                     4, new Simple(1, 4)));

Mobile s5 = new Simple(10, 5);
Mobile s10 = new Simple(8, 10);
Mobile s15 = new Simple(20, 15);

Mobile bs5s10 = new Big(10, 8, this.s5, 4, this.s10);
Mobile bs15bs5s10 = new Big(5, 6, this.s15, 6, this.bs5s10);

Mobile bad1 = new Big(10, 5, this.s5, 5, this.s10);
Mobile bad2 = new Big(5, 6, this.s15, 6, this.bad1);
Mobile bad3 = new Big(5, 6, this.bad1, 6, this.s15);
Mobile bad4 = new Big(5, 3, this.s15, 6, this.bs5s10);

Mobile s5 = new Simple(10, 5);
Mobile s10 = new Simple(8, 10);
```

```
Mobile s15 = new Simple(20, 15);

Mobile bs5s10 = new Big(10, 8, this.s5, 4, this.s10);
Mobile bs15bs5s10 = new Big(5, 6, this.s15, 6, this.bs5s10);

Mobile bad1 = new Big(10, 5, this.s5, 5, this.s10);
Mobile bad2 = new Big(5, 6, this.s15, 6, this.bad1);
Mobile bad3 = new Big(5, 6, this.bad1, 6, this.s15);
Mobile bad4 = new Big(5, 3, this.s15, 6, this.bs5s10);
```

C. Design the method `totalWeight` that computes the total weight of the mobile. Assume that the struts and the cords do not weigh anything.

Solution [POINTS 5: 1 for purpose and header in the interface Mobile, 1 for the body in the Simple class, 1 for the body in the Big class, 2 for examples (must cover both Simple and Big classes, examples for Big must include one with at least two levels).]

```
// in the interface Mobile:
// compute the total weight of a mobile
int totalWeight();

// in the class Simple:
// compute the total weight of a mobile
int totalWeight(){
  return this.weight;  }

// in the class Big:
// compute the total weight of a mobile
int totalWeight(){
  return this.left.totalWeight() + this.right.totalWeight();  }

// in the class Examples:
// test for the method totalWeight
boolean testTotalWeight(){
  return (check this.s5.totalWeight() expect 5) &&
         (check this.s15.totalWeight() expect 15) &&
         (check this.bs5s10.totalWeight() expect 15) &&
         (check this.bs15bs5s10.totalWeight() expect 30);
}
```

15 Points

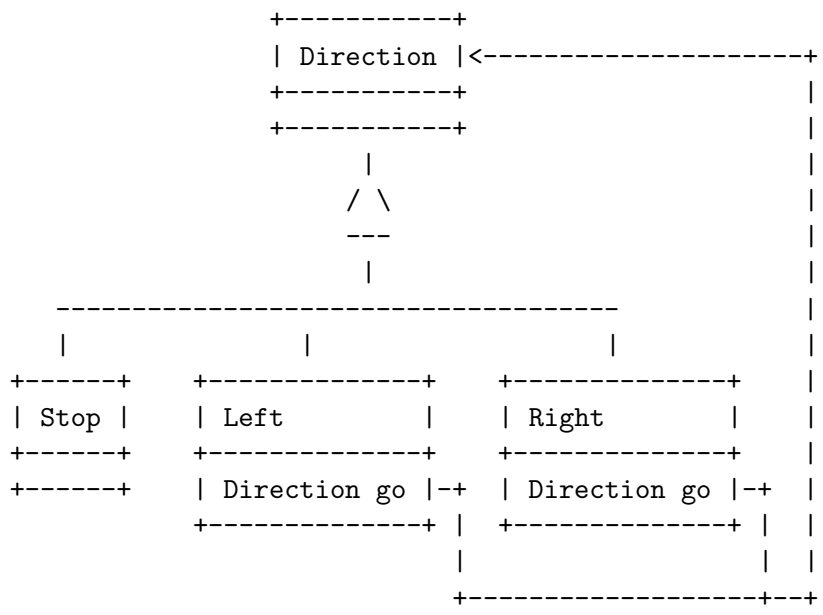
Problem 4

The following classes have been designed to represent some kind of direction:

```
// to represent a direction
interface Direction { }

// to represent the direction to stop
class Stop implements Direction {

  Stop() { }
}

// to represent the direction to go to the left
class Left implements Direction {
  Direction go;

  Left(Direction go) {
    this.go = go; }
}

// to represent the direction to go to the right
class Right implements Direction {
  Direction go;

  Right(Direction go) {
    this.go = go; }
}
```

A. Draw a class diagram for the classes that represent the direction.

Solution [POINTS 3: 1 for the Direction interface, 1 for the three variants, 1 point for the arrows.]



B. Make examples of data that represent the direction.

Solution [POINTS 4: 1 point for an example of Stop, 1 point for examples of Left and Right where 'go' is a Stop, 2 points for more complex examples]

```
Direction stop = new Stop();
Direction left = new Left(this.stop);
Direction right = new Right(this.stop);
Direction lr = new Left(this.right);
Direction rl = new Right(this.left);
Direction lrl = new Left(this.rl);
Direction llrr = new Left(new Left(new Right(new Right(new Stop()))));
```

C. Design the method `countLeft` that counts `Left`s in a direction.

Solution [POINTS 8: 1 point for purpose and header in the Direction interface, 1 point each for the body of the method in the classes Stop, Left, and Right, 4 points for examples (must include Stop, simple Left and Right, direction with several Left-s and Right-s]

```
// in the interface Direction:
// count the number of lefts in this direction
int countLeft();

// in the class Stop:
// count the number of lefts in this direction
int countLeft(){ return 0; }

// in the class Left:
// count the number of lefts in this direction
int countLeft(){ return this.go.countLeft(); }

// in the class Right:
// count the number of lefts in this direction
int countLeft(){ return 1 + this.go.countLeft(); }

// in the class Examples:
// tests for the method countLeft
boolean testCountLeft(){
  return (check this.stop.countLeft() expect 0) &&
         (check this.left.countLeft() expect 1) &&
         (check this.right.countLeft() expect 0) &&
         (check this.lr.countLeft() expect 1) &&
         (check this.rl.countLeft() expect 1) &&
         (check this.lrl.countLeft() expect 2) &&
         (check this.llrr.countLeft() expect 2); }
```