CSU213 Exam 1 – Fall 2006

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		/13
2		/16
3		/10
4		/11
5		/10
Total		/60

13 POINTS

Problem 1

Take a look at these structure and data definitions:

```
// to represent an e-mail message
interface Message {
}

// to represent an email message
class Email implements Message {
  String header;
  String body;

  Email(String header, String body) {
    this.header = header;
    this.body = body;
  }
}

// to represent a forwarded email message
class Forward implements Message {
  String header;
  Message message;

  Forward(String header, Message message) {
    this.header = header;
    this.message = message;
  }
}
```

Answer the following questions in this context:

A. Write down three distinct data examples, at least one per variant:

_____________**Solution** _____________ [POINTS 2: one point per variant]

```
Message msg1 = new Email("vkp", "hello");
Message msg2 = new Email("mf", "bye");

Message msg3 = new Forward("bob", this.msg1);
Message msg4 = new Forward("dan", this.msg3);
Message msg5 = new Forward("ann", this.msg2);
```

B. Write down the template for each class in the `Message` hierarchy:

Solution

[POINTS 3:1 for the template in Email, 1 for the template in Forward, 1 for method invocation with self-reference]

```
class Email implements Message {
 String header;
 String body;
 Email(String header, String body) {
  this.header = header;
  this.body = body;
 }
 ??? method(){
  ... this.header ...  -- String
  ... this.body ... -- String
  ... this.mmm() ...    -- ??
 }
}

class Forward implements Message {
  String header;
  Message message;
  Forward(String header, Message message) {
    this.header = header;
    this.message = message;
  }

 ??? method(){
  ... this.header ...   -- String
  ... this.message ...  -- Message
  ... this.mmm() ...    -- ??
  ... this.message.mm() -- ??
 }
}
```

C. Write down a purpose statement and the header for the method `msgLength` that finds out how long is a message.

____________ **Solution** ____________ [POINTS 1]

```
interface Message {
 // determine the length of this message
 int msgLength();
}
```

D. Write down two method examples (one per variant) for this method.

____________ **Solution** ____________ [POINTS 1]

```
  // test the method msgLength
  boolean testMsgLength =
    (check this.msg1.msgLength() expect 8) &&
    (check this.msg2.msgLength() expect 5) &&
    (check this.msg3.msgLength() expect 11) &&
    (check this.msg4.msgLength() expect 14) &&
    (check this.msg5.msgLength() expect 8);
```

E. Complete the design of the method `msgLength`.

____________ **Solution** ____________ [POINTS 1]

```
interface Message {
 // determine the length of this message
 int msgLength();

 // is this message longer than that given message?
 boolean longerMessage(Message that);
}
class Email implements Message{
 ...

 // is this message longer than that given message?
 boolean longerMessage(Message that){
  return this.msgLength() > that.msgLength();
 }
}
```

```
// .... same in the class Forward ....

  Message msg1 = new Email("vkp", "hello");
  Message msg2 = new Email("mf", "bye");

  Message msg3 = new Forward("bob", this.msg1);
  Message msg4 = new Forward("dan", this.msg3);
  Message msg5 = new Forward("ann", this.msg2);
  Message msg6 = new Email("heisenberg", "good day");

  // test the method longerMessage
  boolean testLongerMessage =
    (check this.msg1.longerMessage(this.msg2) expect true) &&
    (check this.msg2.longerMessage(this.msg1) expect false) &&
    (check this.msg1.longerMessage(this.msg3) expect false) &&
    (check this.msg2.longerMessage(this.msg4) expect false) &&
    (check this.msg6.longerMessage(this.msg4) expect true) &&
    (check this.msg3.longerMessage(this.msg6) expect false) &&
    (check this.msg4.longerMessage(this.msg1) expect true);
```

F. Design the method `sender` that produces the `header` of the original email message.

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

```
interface Message {
 // determine the length of this message
 int msgLength();

 // is this message longer than that given message?
 boolean longerMessage(Message that);

 // to produce the sender of this message
 String sender();
}

 class Email implements Message{
  ...

  // to produce the sender of this message
  String sender(){
    return this.header;
  }
}

class Forward implements Message {
 ...

  // to produce the sender of this message
  String sender(){
    return this.message.sender();
  }
}

  // Eaxmples:
  Message msg1 = new Email("vkp", "hello");
  Message msg2 = new Email("mf", "bye");

  Message msg3 = new Forward("bob", this.msg1);
  Message msg4 = new Forward("dan", this.msg3);
```

```
Message msg5 = new Forward("ann", this.msg2);
Message msg6 = new Email("heisenberg", "good day");

// test the method sender
boolean testSender =
  (check this.msg1.sender() expect "vkp") &&
  (check this.msg2.sender() expect "mf") &&
  (check this.msg3.sender() expect "vkp") &&
  (check this.msg4.sender() expect "vkp") &&
  (check this.msg5.sender() expect "mf");
```

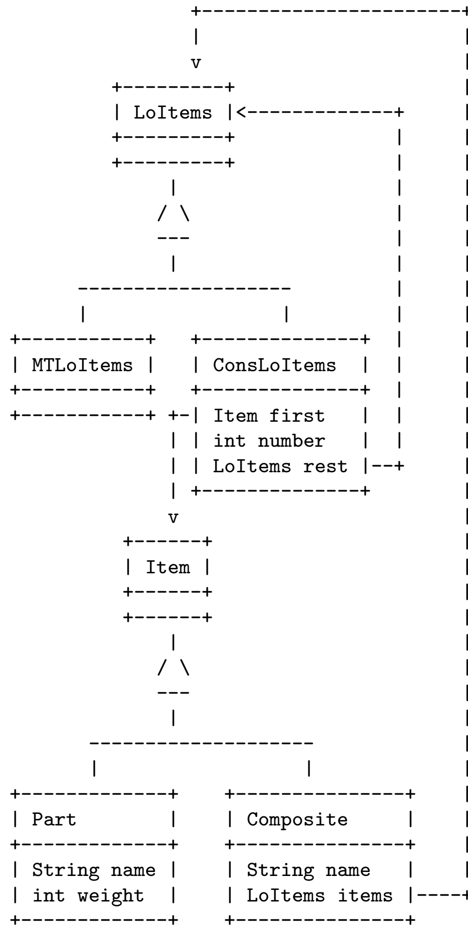
16 Points

Problem 2

The main purpose of this exercise is to develop the method `totalWeight` of an item in the manufacturer's database.

The factory manufacturing keeps track of the parts needed to produce different items. An item may be composed of several parts or of already assembled composite items. For example, a car wheel consists of the rim and the tire, (and other parts...), and we need four wheels to build a car. We also need two doors, each consisting of a frame, a glass pane, and a lock.

Here are the relevant class definitions:



Note, that when constructing the list of items needed to build one composite item, if there are three identical items needed, we can just include the needed item once and specify that we need three of them, instead of repeating the same item three times in the list.

A. Represent the example of a car that was given in the introduction to this problem as data in this database.

______ **Solution** ______ [POINTS 3: 1 for the four parts, 1 for the wheel and the door, 1 for the auto, 1 if the number field is used to represent the multiplicity]

```
Item tire = new Part("Tire", 40);
Item rim = new Part("Rim", 20);

Item wheel = new Composite("wheel", new ConsLoItems(this.tire, 1,
                                    new ConsLoItems(this.rim, 1,
                                    new MTLoItems())));

Item glass = new Part("glass", 10);
Item frame = new Part("frame", 20);
Item lock = new Part("lock", 5);

Item door = new Composite("door", new ConsLoItems(this.glass, 1,
                                  new ConsLoItems(this.frame, 1,
                                  new ConsLoItems(this.lock, 1,
                                  new MTLoItems()))));

Item auto = new Composite("auto", new ConsLoItems(this.door, 2,
                                  new ConsLoItems(this.wheel, 4,
                                  new MTLoItems())));
```

B. Show the template for each of the classes in this data definition.

______________**Solution**______________ [POINTS 4: 2 for ConsLoItems – inlcuding method for 'first' and self-reference, 2 for Composite]

```
// in the class ConsLoItems:
/* TEMPLATE
  ... this.first ...                  -- Item
  ... this.first.methodName() ...     -- returnType
  ... this.number ...                 -- int
  ... this.rest ...                   -- LoItems
  ... this.rest.methodName() ...      -- returnType
*/

// in the class Item:
/* TEMPLATE
  ... this.name ...                 -- String
  ... this.weight ...               -- int
*/

// in the class Composite:
/* TEMPLATE
  ... this.name ...                 -- String
  ... this.items ...                -- LoItems
  ... this.items.methodName() ...   -- returnType
*/
```

C. Design the method `totalWeight` that computes the total weight of an item in the manufacturer's database.

____________ **Solution** ____________ [POINTS 9: 4 for the 4 bodies of the methods, 4 for examples for 4 methods, 1 points for method declarations in the 1 interfaces

```
// interface LoItems:
  // produce the total weight of this list of items
  int totalWeight();
...

// class MTLoItems implements LoItems:
  // produce the total weight of this list of items
  int totalWeight(){ return 0; }

// class ConsLoItems implements LoItems:
  // produce the total weight of this list of items
  int totalWeight(){
    return this.first.totalWeight() * this.number +
           this.rest.totalWeight();
  }

// interface Item:
  // produce the total weight of this item
  int totalWeight();

// class Part implements Item:
  // produce the total weight of this item
  int totalWeight(){ return this.weight; }

// class Composite implements Item:
  // produce the total weight of this item
  int totalWeight(){
    return this.items.totalWeight();
  }

// class Examples:
  Item tire = new Part("Tire", 40);
  Item rim = new Part("Rim", 20);
```

```
Item wheel = new Composite("wheel", new ConsLoItems(this.tire, 1,
                                    new ConsLoItems(this.rim, 1,
                                    new MTLoItems())));

LoItems wheellist = new ConsLoItems(this.tire, 1,
                    new ConsLoItems(this.rim, 1,
                    new MTLoItems()));

Item glass = new Part("glass", 10);
Item frame = new Part("frame", 20);
Item lock = new Part("lock", 5);

Item door = new Composite("door", new ConsLoItems(this.glass, 1,
                                  new ConsLoItems(this.frame, 1,
                                  new ConsLoItems(this.lock, 1,
                                  new MTLoItems()))));

LoItems doorlist = new ConsLoItems(this.glass, 1,
                   new ConsLoItems(this.frame, 1,
                   new ConsLoItems(this.lock, 1,
                   new MTLoItems())));


Item auto = new Composite("auto", new ConsLoItems(this.door, 2,
                                  new ConsLoItems(this.wheel, 4,
                                  new MTLoItems())));

// test the method totalWeight in the class Part
boolean testTotalWeightPart =
  (check this.tire.totalWeight() expect 40) &&
  (check this.lock.totalWeight() expect 5);

// test the method totalWeight in the class LoItems
boolean testTotalWeightLoItems =
  (check (new MTLoItems()).totalWeight() expect 0) &&
  (check this.wheellist.totalWeight() expect 60) &&
  (check this.doorlist.totalWeight() expect 35);

// test the method totalWeight in the class Composite
boolean testTotalWeightComposite =
```

```
(check this.auto.totalWeight() expect 310);
```

10 Points

Problem 3

This and the next problem deal with magic colors. There are only three magic colors: red, green, and blue. The magic is in the way that the colors mix. Adding any color to itself produces the same color. Adding any color other than red to red produces green, adding any color other than green to green produces blue, and adding any color other than blue to blue produces red.

We decided to represent the colors as follows:

```
// represent a magic color
class MagicColor{
 String col;
 MagicColor(String col) {
  this.col = col;
 }
}
```

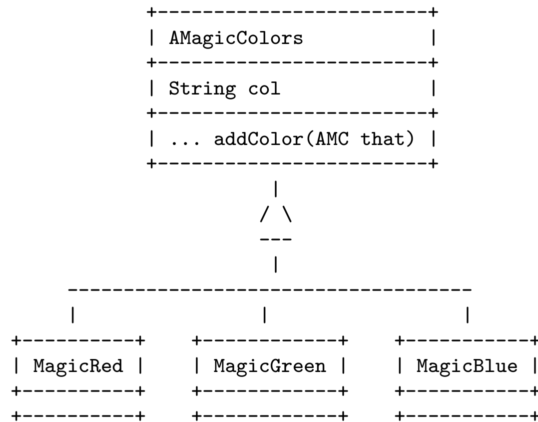
A. Design the method `addColor` that implements the adding of the colors as explained above.

Solution

```
;; [POINTS 3, ......]
// produce the magic color you get by adding the given color to this one
MagicColor addColor(MagicColor that){
...
}
```

B. We do not like the repetition in the methods. We think that representing the magic colors by derived subclasses with an abstract super class would work better.

Our new class hierarchy begins to look as follows:



Convert your solution to work with this data representation as follows:

(a) Design the constructors - both in the `super` class and in the subclasses.

(b) Convert your examples of data from the *Part A* to data in this new representation.

Solution

```
;; [POINTS 3, ......]
// produce the magic color you get by adding the given color to this one
MagicColor addColor(MagicColor that){
...
}
```

(c) Design the method `addColor`. Avoid as much repetition as possible. *Follow the Design Recipe.*

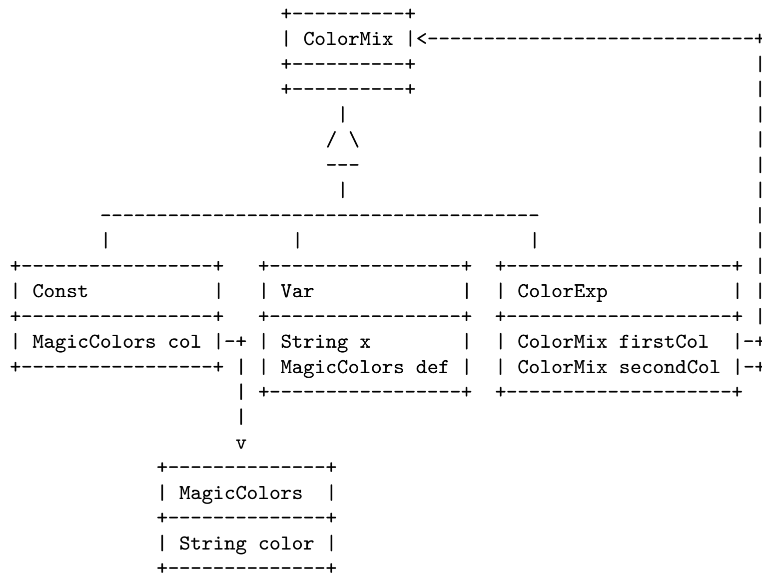
Solution

```
;; [POINTS 3, ......]
// produce the magic color you get by adding the given color to this one
MagicColor addColor(MagicColor that){
...
}
```

11 Points

Problem 4

We now want to keep mixing magic colors and explore what colors we get as the result. For this we designed a *magic color arithmetic* that allows us to express different color combinations. The following class diagram represents our color combinations arithmetic:



In the class `Var` we can substitute any color for the variable represented by the `String s`. If no substitutions are given, we assume that the *d*efault value is to be used in determining the final color of the color mixing expression.

A. Design the method `eval` that determines the final color after the colors have been mixed in the manner corresponding to the color mix. Use the `def` (default color) in the class `Var`. ——————**Solution** ——————

```
/*
[POINTS 2]
Here comes my solution

*/
```

B. Design the method `subst` that consumes a `String s` and an instance of `MagicColors col` and replaces each variable expression `Var` that contains the given string by a constant expression `Const` with the value given by the `MagicColors col`.

Do not inlcude the field definitions and the constructors, but make it clear in which class each method is defined. ______________ **Solution**

____

[POINTS 2: data examples]

. . . . . .
. . . . . .

10 Points

Problem 5

On a road trip across the country we record at each stop the current odometer reading and the number of galons of gas we purchased. We are iterested in our miles-per-gallon consumption, and also what was our best 'gas mileage'. The following classes represent a record of our trip:

```
// to represent one stop on a road trip
class Stop {
  double mile;
  double gallons;

  Stop(double mile, double gallons) {
    this.mile = mile;
    this.gallons = gallons;
  }
}

// to represent a car trip
interface Trip {
}

// to represent the start of a trip
class Start implements Trip {
  Stop stop;

  Start(Stop stop) {
    this.stop = stop;
  }
}

// to represent an actual car trip
class ConsTrip implements Trip {
  Stop first;
  Trip rest;

  ConsTrip(Stop first, Trip rest) {
    this.first = first;
    this.rest = rest;
  }
}
```

A. Design the method `mpg` that computes the number of miles we traveled per gallon of gas on the part of the trip from one stop to another.

Solution

[POINTS: 10: 4 helper method, 3 examples, 3 main method]

```
// to represent a trip of one or more segments
interface Trip {
smaller(s2);  // should be s1

}
```

B. Design the method `bestMPG` that computes the best (highest) *mpg* over the whole trip.

Solution

[POINTS: 10: 4 helper method, 3 examples, 3 main method]

```
// to represent a trip of one or more segments
interface Trip {
smaller(s2);  // should be s1

}
```