CS 2510 Exam 2 – Fall 2010

Name: _____

Student Id (last 4 digits): _____

• Write down the answers in the space provided.
• You may use any features of standard Java, except the `instanceof` operator. If you need a method and you don't know whether it is provided, you must define it. You do not need to include the curly braces for every `if` or every `else`, as long as the statements you write are otherwise correct.
• 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.

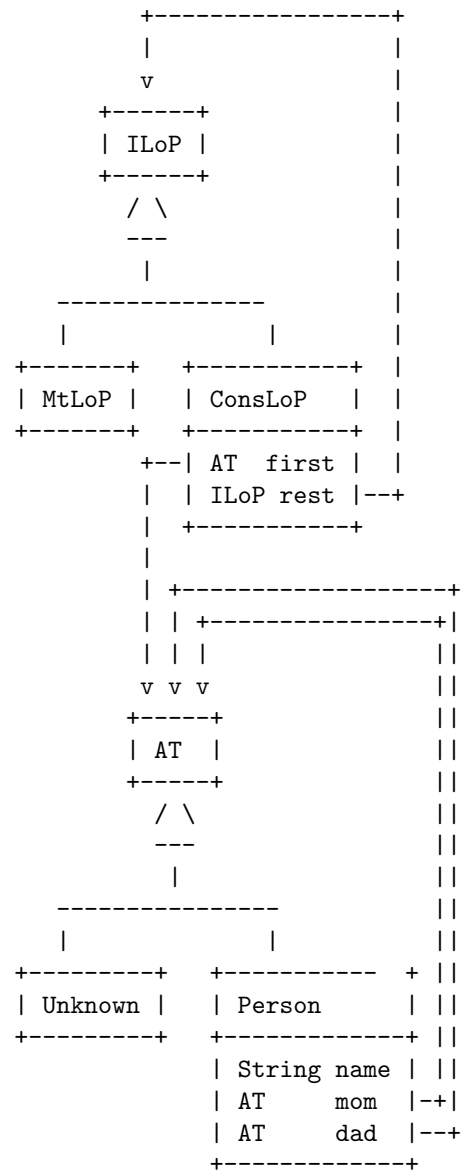
Good luck.

Problem	Points	/
A		/ 0
B		/ 4
C		/ 8
D		/ 8
D		/ 4
Total		/26

25 Points

Problem 1

Here is a Java class diagram that describes the information we have about a number of persons and their ancestors:



A. *(0 points)*

You are not required to do this. You may want to see what the data definitions look like. Do not spend much time on this, unless you do not understand what the data represents. The examples in the next part should make that clear.

Write down the Java class and interface definitions that are represented by this class diagram.

Solution [POINTS 0:]

```
// to represent a list of persons
interface ILoP{  }

// to represent an empty list of persons
class MtLoP implements ILoP{
  MtLoP(){}
}

// to represent a nonempty list of persons
class ConsLoP implements ILoP{
  AT first;
  ILoP rest;

  ConsLoP(AT first, ILoP rest){
    this.first = first;
    this.rest = rest;
  }
}

// to represent an ancestor tree of a person, or an unknown ancestor
interface AT{ }

// to represent an unknown ancestor in an ancestor tree
class Unknown implements AT{
  Unknown(){}
}

// to represent a person in an ancestor tree
class Person implements AT{
  String name;
```

```
  AT mom;
  AT dad;

  Person(String name, AT mom, AT dad){
    this.name = name;
    this.mom = mom;
    this.dad = dad;
  }
}
```

B. *(4 points)*

Make examples of the data, if you know the following about the various people:

We have information about Ann, Cal, Dan, Eli, Fay, Jon, Kim, Liz, May, Pat, Ron, Sam, Tom, Val, and Zoe. (You may use just the first initials instead of the full names.)

Ann's mom is May, Ann's dad is Sam Dan's mom is Fay, Dan's dad is Ron Eli's mom is Liz, Eli's dad is Tom Kim's mom is Pat, Kim's dad is Cal May's dad is Jon Pat's mom is May, Pat's dad is Eli Sam's mom is Val, Sam's dad is Ron Zoe's mom is May, Zoe's dad is Sam

We do not know anything about the remaining ancestors.

Solution [POINTS 4: 1 point for the class `Goal`, 1 point for the class `Tunnel`, 2 points for the class `Room`]

```
AT xx = new Unknown();

AT cal = new Person("Cal", this.xx, this.xx);
AT fay = new Person("Fay", this.xx, this.xx);
AT jon = new Person("Jon", this.xx, this.xx);
AT liz = new Person("Liz", this.xx, this.xx);
AT ron = new Person("Ron", this.xx, this.xx);
AT tom = new Person("Tom", this.xx, this.xx);
AT val = new Person("Val", this.xx, this.xx);

AT dan = new Person("Dan", this.fay, this.ron);
AT eli = new Person("Eli", this.liz, this.tom);
AT may = new Person("May", this.xx, this.jon);
AT sam = new Person("Sam", this.val, this.ron);

AT ann = new Person("Ann", this.may, this.sam);
AT pat = new Person("Pat", this.may, this.eli);
AT zoe = new Person("Zoe", this.may, this.sam);

AT kim = new Person("Kim", this.pat, this.cal);

ILoP mtlop = new MtLoP();

ILoP plist =
  new ConsLoP(this.ann,
  new ConsLoP(this.cal,
  new ConsLoP(this.dan,
```

```
new ConsLoP(this.eli,
new ConsLoP(this.fay,
new ConsLoP(this.jon,
new ConsLoP(this.kim,
new ConsLoP(this.liz,
new ConsLoP(this.may,
new ConsLoP(this.pat,
new ConsLoP(this.ron,
new ConsLoP(this.sam,
new ConsLoP(this.tom,
new ConsLoP(this.val,
new ConsLoP(this.zoe, this.mtlop))))))))))));
```

C. *(8 points)* Design the method `ancestorList` that produces a list (`ILoP` of all `Person`s that are ancestors of a person. Do not include in the list the *unknown*s, but do include the person himself/herself.

Solution [POINTS 8: (ancestorList: 1 point purpose/header; 2 points bodies in Person and AT/Unknown, 2 points tests); (append method for ILoP: 1 point purpose/header; 1 point bodies; 1 point tests)]

```
// in the interface ILoP:
  // append the given list to this one
  public ILoP append(ILoP that);

// in the class MtLoP:
  // append the given list to this one
  public ILoP append(ILoP that){
    return that;
  }

// in the class ConsLoP:
  // append the given list to this one
  public ILoP append(ILoP that){
    return new ConsLoP(this.first, this.rest.append(that));
  }


// in the class Examples:
  // test the method valueOf for the maze classes
  boolean testValueOf(Tester t){
    return
    t.checkExpect(this.gold.valueOf("Gold"), 20) &&
    t.checkExpect(this.t2.valueOf("Gold"), 20) &&
    t.checkExpect(this.t1.valueOf("Gold"), 0) &&
    t.checkExpect(this.r0.valueOf("Gold"), 50);
  }

in the interface AT:
  //produce a list of ancestors for this person
  public ILoP ancestorList();
```

```
in the class Unknown:
  //produce a list of ancestors for this person
  public ILoP ancestorList(){
    return new MtLoP();
  }

in the class Person:
  // produce a list of ancestors for this person
  public ILoP ancestorList(){
    return new ConsLoP(this,
        this.mom.ancestorList().append(this.dad.ancestorList()));
  }

in the class Examples:
    t.checkExpect(this.cal.ancestorList(),
        new ConsLoP(this.cal, this.mtlop));

    t.checkExpect(this.eli.ancestorList(),
        new ConsLoP(this.eli,
            new ConsLoP(this.liz,
                new ConsLoP(this.tom, this.mtlop))));

    t.checkExpect(this.pat.ancestorList(),
        new ConsLoP(this.pat,
            new ConsLoP(this.may,
                new ConsLoP(this.jon,
                    new ConsLoP(this.eli,
                        new ConsLoP(this.liz,
                            new ConsLoP(this.tom, this.mtlop)))))));
```

D. *(8 points)*

Design the method `hasCommon` that determines whether two persons (*this and the given one*) have a common ancestor.

Solution [POINTS 8: (3 points for commonAncestor in AT...): 1 point purpose/header; 1 point body for the `Person` class, 1 point examples;

(3 points for hasCommon in ILoP...): 1 point purpose/header; 1 point body for the `Cons` class, 1 point examples;

(2 points for isInList in AT ...): 1 point purpose and body; 1 point examples]

```
in the interface AT:
  // does this and the given person have a common ancestor
  public boolean commonAncestor(AT that);

  // is this person in the given list of people -
  // not counting Unknowns
  public boolean inInList(ILoP that);

in the class Unknown:
  // does this and the given person have a common ancestor
  public boolean commonAncestor(AT that){
    return false;
  }

  // is this person in the given list of people -
  // not counting Unknowns
  public boolean inInList(ILoP that){
    return false;
  }

// in the class Person:
  // does this and the given person have a common ancestor
  public boolean commonAncestor(AT that){
    return this.ancestorList().hasCommon(that.ancestorList());
  }

  // is this person in the given list of people -
  // not counting Unknowns
  public boolean inInList(ILoP that){
    return that.contains(this.name);
  }
```

```
// in the interface ILoP:
  // does this list and the given one contain a common person?
  public boolean hasCommon(ILoP that);

// in the class MtLoP:
  // does this list and the given one contain a common person?
  public boolean hasCommon(ILoP that){
    return false;
  }

// in the class ConsLoP:
  // does this list and the given one contain a common person?
  public boolean hasCommon(ILoP that){
    return
    this.first.inInList(that) ||
    this.rest.hasCommon(that);
  }

// in the class Examples:
  void testIsInList(Tester t){
    t.checkExpect(this.ann.isInList(this.list1), true);
    t.checkExpect(this.ann.isInList(this.list2), false);
    t.checkExpect(this.ann.isInList(this.mtlop), false);
  }

  void testHasCommon(Tester t){
    t.checkExpect(this.list1a.hasCommon(this.mtlop), false);
    t.checkExpect(this.mtlop.hasCommon(this.list2), false);
    t.checkExpect(this.list1a.hasCommon(this.list2), false);
    t.checkExpect(this.list12.hasCommon(this.plist), true);
  }

  void testCommonAncestor(Tester t){
    t.checkExpect(this.pat.commonAncestor(this.zoe), true);
  }
```

... This page is intentionally left blank ...

E. *(5 points)*

Show the templates for all classes in this problem for which you have designed methods.

Solution [POINTS 5: 2 points template for `ConsLoP`; 2 points template for `Person`; 1 point for all other templates]

```
// in the class Goal
TEMPLATE:
  FIELDS:
  ... this.prize ...              -- String
  ... this.value ...              -- int

  METHODS:
  ... this.valueOf(String) ...    -- int
  ... this.longestPath() ...      -- int

  METHODS FOR FIELDS:


// in the class Tunnel
 TEMPLATE:
  FIELDS:
  ... this.next ...               -- IMaze
  ... this.length ...             -- int

  METHODS:
  ... this.valueOf(String) ...    -- int
  ... this.longestPath() ...      -- int

  METHODS FOR FIELDS:
  ... this.next.valueOf(String) ...    -- int
  ... this.next.longestPath() ...      -- int


// in the class Room
 TEMPLATE:
  FIELDS:
  ... this.left ...               -- IMaze
```

```
... this.right ...              -- IMaze

METHODS:
... this.valueOf(String) ...    -- int
... this.longestPath() ...      -- int

METHODS FOR FIELDS:
... this.left.valueOf(String) ...    -- int
... this.left.longestPath() ...      -- int

... this.right.valueOf(String) ...    -- int
... this.right.longestPath() ...      -- int
```