Write down the answers in the space provided.

You may use all syntax that you know from FunJava other than abstract classes. If you need a method and you don’t know whether it is provided, 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 in FunJava.

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 1

Here is a Java class diagram that describes a very cold person covered with a pile of blankets:

```
+-------------------------+
| +---------------------+ |
| IPile |                |
+-------+                |
<table>
<thead>
<tr>
<th>/ \</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>-------</td>
</tr>
</tbody>
</table>
+---------+ +---------+ +---------+ +---------+ +---------+
| Bottom | | Top | | | String name | | String color | |
+---------+ +---------+ | IPile pile | ---+
```

25 Points
A. (2 points)

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

Solution [POINTS 3: 1 point for the interface, 1 point for the class Bottom, 1 point for the class Top]

```java
// to represent a pile of blankets with someone below
interface IPile {
}

// to represent a person on the bottom of the pile
class Bottom implements IPile {
    String name;
    Bottom(String name) {
        this.name = name;
    }
}

// to represents a pile of blankets atop a person
class Top implements IPile {
    String color;
    IPile pile;
    Top(String color, IPile pile) {
        this.color = color;
        this.pile = pile;
    }
}
```
B. \( (2 \text{ points}) \)

Make examples of three piles - one without anything on top, plus two other piles, one of which must have at least two Tops.

\[\text{Solution} \quad [\text{POINTS 2}: \text{one point for the Bottom, and one point for the example with two Tops.}]\]

\[
\text{IPile pat = new Bottom("Pat");} \\
\text{IPile kim = new Bottom("Kim");} \\
\text{IPile patPile =} \\
\quad \text{new Top("red", new Top("blue", new Top("green", this.pat))});} \\
\text{IPile kimPile =} \\
\quad \text{new Top("green", new Top("black", this.kim))};} \\
\text{IPile bigPile = new Top("red", new Top("yellow",} \\
\quad \quad \text{new Top("blue", new Top("yellow", new Top("red", this.pat))))});}
\]
C. (5 points) Design the method hasColor that determines whether a pile includes a top of a given color.

Solution [POINTS 5: 1 point purpose/header; 1 point body in each class; 2 points examples – should include result 0 and result > 1]

// in the interface IPile:
// is there a given color in this pile?
boolean hasColor(String color);

// in the class Bottom:
// is there a given color in this pile?
boolean hasColor(String color){
    return false;
}

// in the class Top:
/* TEMPLATE:
... this.color ... -- String
... this.pile ... -- IPile

... this.pile.hasColor(String) ... -- boolean */

// is there a given color in this pile?
boolean hasColor(String color){
    return this.color.equals(color) ||
           this.pile.hasColor(color);
}

// in the class Examples:
// test the method hasColor
boolean testHasColor(Tester t){
    return
    t.checkExpect(this.pat.hasColor("red"), false) &&
    t.checkExpect(this.patPile.hasColor("red"), true) &&
    t.checkExpect(this.kimPile.hasColor("red"), false) &&
    t.checkExpect(this.bigPile.hasColor("red"), true);
}
D. (8 points)

Looking at the two piles you wonder which one is bigger. Design the method `isBigger` that determines whether one pile is bigger than another one.

****Solution**** [POINTS 8: 1 point purpose/header; 1 point body for the `Bottom` class, 1 point body for the `Top` class, 3 points for definition of the helper method (purpose + header; body; examples/tests), 2 points for examples for the `isBigger` method.]

```java
// in the interface IPile:
// is this pile bigger than the given one?
boolean isBigger(IPile that);

// compute the size of this pile
int pileSize();

// in the class Bottom:
// is this pile bigger than the given one?
boolean isBigger(IPile that){
    return false;
}

// compute the size of this pile
int pileSize(){
    return 0;
}

// in the class Top:
// is this pile bigger than the given one?
boolean isBigger(IPile that){
    return this.pileSize() > that.pileSize();
}

// compute the size of this pile
int pileSize(){
    return 1 + this.pile.pileSize();
}

// in the class Examples:
// test the method isBigger
boolean testIsBigger(Tester t){
    return 
        t.checkExpect(this.pat.isBigger(this.kim), false) &&
        t.checkExpect(this.pat.isBigger(this.kimPile), false) &&
```
t.checkExpect(this.patPile.isBigger(this.kim), true) &&
t.checkExpect(this.kimPile.isBigger(this.patPile), false) &&
t.checkExpect(this.bigPile.isBigger(this.kimPile), true);
}

// test the method pileSize
boolean testPileSize(Tester t){
    return
    t.checkExpect(this.pat.pileSize(), 0) &&
    t.checkExpect(this.patPile.pileSize(), 3) &&
    t.checkExpect(this.kimPile.pileSize(), 2) &&
    t.checkExpect(this.bigPile.pileSize(), 5);
}
E. (6 points)

Now you wonder whether two piles have any blankets of the same color. Design the method `commonColor` that determines whether two piles share at least one color of blankets.

**Solution**

[POINTS 6: 1 point purpose/header; 1 point body for the `Bottom` class, 2 points body for the `Top` class (should invoke `that.hasColor`, 2 points for examples for the `commonColor` method.]

// in the interface IPile:
// does this pile and the given pile have a color in common?
boolean commonColor(IPile that);

// in the class Bottom:
// is there a given color in this pile?
boolean hasColor(String color){
    return false;
}

// in the class Top:
// does this pile and the given pile have a color in common?
boolean commonColor(IPile that){
    return that.hasColor(this.color) ||
           this.pile.commonColor(that);
}

// in the class Examples:
// test the method commonColor
boolean testCommonColor(Tester t){
    return
t.checkExpect(this.pat.commonColor(this.kim), false) &
t.checkExpect(this.pat.commonColor(this.kimPile), false) &
t.checkExpect(this.patPile.commonColor(this.kim), false) &
t.checkExpect(this.kimPile.commonColor(this.patPile), true) &
t.checkExpect(this.bigPile.commonColor(this.kimPile), false) &
t.checkExpect(this.bigPile.commonColor(this.patPile), true);
F. (4 points)

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

Solution [POINTS 4: 1 point template for Bottom, 3 points template for Top: 1 point for fields, 1 point for methods for contents, 1 point for data types]

// in the class Bottom
TEMPLATE:
FIELDS:
... this.name ... -- String

METHODS:
... this.hasColor(String) ... -- boolean
... this.isBigger(IPile) ... -- boolean
... this.pileSize() ... -- int
... this.commonColor(IPile) ... -- boolean

METHODS FOR FIELDS:

// in the class Top
TEMPLATE:
FIELDS:
... this.color ... -- String
... this.pile ... -- IPile

METHODS:
... this.hasColor(String) ... -- boolean
... this.isBigger(IPile) ... -- boolean
... this.pileSize() ... -- int
... this.commonColor(IPile) ... -- boolean

METHODS FOR FIELDS:
... this.pile.hasColor(String) ... -- boolean
... this.pile.isBigger(IPile) ... -- boolean
... this.pile.pileSize() ... -- int
... this.pile.commonColor(IPile) ... -- boolean