CS 2510 Exam 1 – Spring 2010

Name: ____________________________

Student Id (last 4 digits): ____________________________

• 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	Points	/
1		/27
Total		/27

25 Points

Problem 1

Here is a Java class diagram that describes a pizza order with the selection of the crust and toppings:

```
           +-------------------------+
           | +---------------------+ |
           v v                     | |
        +--------+                 | |
        | IPizza |                 | |
        +--------+                 | |
           / \                     | |
           ---                     | |
            |                      | |
     -------------------           | |
     |                 |           | |
+-------------+ +----------------+ | |
| Crust       | | Topping        | | |
+-------------+ +----------------+ | |
| String name | | String flavor  |-+ |
+-------------+ | IPizza pizza   |---+
                +----------------+
```

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 `Crust`, 1 point for the class `Topping`]

```
// to represent a pizza order
interface IPizza{ }

// to represent the crust in a pizza order
class Crust implements IPizza{
  String name;

  Crust(String name){
    this.name = name;
  }
}

// to represent the pizza with all its toppings
class Topping implements IPizza{
  String flavor;
  IPizza pizza;

  Topping(String flavor, IPizza pizza){
    this.flavor = flavor;
    this.pizza = pizza;
  }
}
```

B. *(2 points)*

Make examples of three pizza orders - one plain, plus two other pizzas, one of which must have at least two `Topping`s.

____________**Solution** ____________ [POINTS 2: one point for the `Crust`, and one point for the example with two `Topping`s.]

```
IPizza thin = new Crust("thin");
IPizza deepDish = new Crust("deepdish");

IPizza thinPizza =
  new Topping("onion",
      new Topping("pepper",
          new Topping("olives", this.thin)));
IPizza deepDishPizza =
  new Topping("olives",
      new Topping("black", this.deepDish));
IPizza bigpizza =
  new Topping("onion",
      new Topping("mushrooms",
          new Topping("pepper",
              new Topping("mushrooms",
                  new Topping("onion", this.thin)))));
```

C. *(5 points)* Design the method `hasCrust` that determines whether the pizza crust is the kind we are thinking of.

______________**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 IPizza:
  // does this pizza have the given crust?
  boolean hasCrust(String crust);

// in the class Crust:
  // does this pizza have the given crust?
  boolean hasCrust(String crust){
    return this.name.equals(crust);
  }

// in the class Topping:
  /* TEMPLATE:
    ... this.tname ...                -- String
    ... this.pizza ...                -- IPizza

    ... this.pizza.hasCrust(String) ...  -- boolean
  */

  // does this pizza have the given crust?
  boolean hasCrust(String crust){
    return this.pizza.hasCrust(crust);
  }

// in the class Examples:
  // test the method hasCrust
  boolean testHasCrust(Tester t){
    return
    t.checkExpect(this.thin.hasCrust("deepDish"), false) &&
    t.checkExpect(this.thinPizza.hasCrust("thin"), true) &&
    t.checkExpect(this.deepDishPizza.hasCrust("thin"), false) &&
    t.checkExpect(this.bigpizza.hasCrust("thin"), true);
  }
```

D. *(8 points)*

Looking at the two pizzas you wonder which one cost more. Design the method costsMore that determines whether one pizza costs more than another one. The price for plain pizza is $10, each topping costs $1.

____________**Solution**____________ [POINTS 8: 1 point purpose/header; 1 point body for the Crust class, 1 point body for the Topping class, 3 points for definition of the helper method(purpose + header; body; examples/tests), 2 points for examples for the costsMore method.]

```
// in the interface IPizza:
  // does this pizza cost more than the given one?
  boolean costsMore(IPizza that);

  // compute the cost of this pizza
  int pizzaCost();

// in the class Crust:
  // does this pizza cost more than the given one?
  boolean costsMore(IPizza that){
    return this.pizzaCost() > that.pizzaCost();
  }

  // compute the cost of this pizza
  int pizzaCost(){
    return 1 + this.pizza.pizzaCost();
  }

// in the class Topping:
  // does this pizza cost more than the given one?
  boolean costsMore(IPizza that){
    return this.pizzaCost() > that.pizzaCost();
  }

  // compute the cost of this pizza
  int pizzaCost(){
    return 1 + this.pizza.pizzaCost();
  }

// in the class Examples:
  // test the method costsMore
  boolean testCostsMore(Tester t){
    return
    t.checkExpect(this.thin.costsMore(this.deepDish), false) &&
    t.checkExpect(this.thin.costsMore(this.deepDishPizza), false) &&
    t.checkExpect(this.thinPizza.costsMore(this.deepDish), true) &&
    t.checkExpect(this.deepDishPizza.costsMore(this.thinPizza), false) &&
    t.checkExpect(this.bigpizza.costsMore(this.deepDishPizza), true);
  }

  // test the method pizzaCost
  boolean testPizzaCost(Tester t){
    return
    t.checkExpect(this.thin.pizzaCost(), 10) &&
    t.checkExpect(this.thinPizza.pizzaCost(), 13) &&
    t.checkExpect(this.deepDishPizza.pizzaCost(), 12) &&
    t.checkExpect(this.bigpizza.pizzaCost(), 15);
  }
```

E. *(6 points)*

Now you wonder whether two pizzas have the same crust. Design the method `sameCrust` that determines whether two pizzas have the same crust.

______________**Solution** ______________ [POINTS 6: 1 point purpose/header; 1 point body for the `Crust` class, 2 points body for the `Topping` class (should invoke `that.hasCrust`, 2 points for examples for the `sameCrust` method.]

```
// in the interface IPizza:
  // does this pizza and the given pizza have the same crust?
  boolean sameCrust(IPizza that);

// in the class Crust:
  // does this pizza and the given pizza have the same crust?
  boolean sameCrust(IPizza that){
    return that.hasCrust(this.name);
  }

// in the class Topping:
  // does this pizza and the given pizza have the same crust?
  boolean sameCrust(IPizza that){
    return this.pizza.sameCrust(that);
  }

// in the class Examples:
  // test the method sameCrust
  boolean testSameCrust(Tester t){
    return
    t.checkExpect(this.thin.sameCrust(this.deepDish), false) &&
    t.checkExpect(this.thin.sameCrust(this.deepDishPizza), false) &&
    t.checkExpect(this.thinPizza.sameCrust(this.deepDish), false) &&
    t.checkExpect(this.deepDishPizza.sameCrust(this.thinPizza), false) &&
    t.checkExpect(this.bigpizza.sameCrust(this.deepDishPizza), false) &&
    t.checkExpect(this.bigpizza.sameCrust(this.thin), 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 Crust
   TEMPLATE:
    FIELDS:
    ... this.name ...                -- String

    METHODS:
    ... this.hasCrust(String) ...    -- boolean
    ... this.costsMore(IPizza) ...   -- boolean
    ... this.pizzaCost() ...         -- int
    ... this.sameCrust(IPizza) ...   -- boolean

    METHODS FOR FIELDS:


// in the class Topping
 TEMPLATE:
  FIELDS:
  ... this.flavor ...                -- String
  ... this.pizza ...                 -- IPizza

  METHODS:
  ... this.hasCrust(String) ...       -- boolean
  ... this.costsMore(IPizza) ...      -- boolean
  ... this.pizzaCost() ...            -- int
  ... this.sameCrust(IPizza) ...      -- boolean

  METHODS FOR FIELDS:
  ... this.pizza.hasCrust(String) ...      -- boolean
  ... this.pizza.costsMore(IPizza) ...     -- boolean
  ... this.pizza.pizzaCost() ...           -- int
  ... this.pizza.sameCrust(IPizza) ...     -- boolean
```