Your fellow designer Andrea presents the design (fragment) above.

(a) (6 points) Which pattern do you recognize?
- Adapter
- Template
- Strategy
- Decorator

(b) (6 points) Which pattern would you prefer for this case?
- Adapter
- Template
- Strategy
- Decorator

(c) (10 points) Redraw the diagram above according to your preferred pattern. State clearly which classes are abstract.

"component"

"Pizza"

"Topping"
- myComp: Component

- Regular
- Italian
- Pan
- anch
- pepper
- cheese

"..." = abstract
Now suppose that we need to calculate the price of a custom-made pizza. Here is a fragment of the price calculation:

<table>
<thead>
<tr>
<th></th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular pizza (without toppings)</td>
<td>5.00</td>
</tr>
<tr>
<td>pan pizza (without toppings)</td>
<td>6.00</td>
</tr>
<tr>
<td>Italian pizza (without toppings)</td>
<td>6.00</td>
</tr>
<tr>
<td>pepperoni</td>
<td>1.00 extra</td>
</tr>
<tr>
<td>anchovies</td>
<td>1.00 extra</td>
</tr>
<tr>
<td>cheese</td>
<td>0.50 extra</td>
</tr>
</tbody>
</table>

For the price calculation, we add the following method to the class Regular.

```java
public float price () { return 5.00; }
```

(d) (8 points) Write two more code fragments for the body of `price()`, one for the class `Italian` and one for the class `Anchovies`. There are only two kinds of bodies possible: `{ return ...; }` or `{ return ... + ...; }

```java
// Italian

{ return 6.00; }

// Anchovies

{ return 1.00 + myComp.price(); }
```
Question 3: Observer (26 points)

Have a look of the abstract interface part of the Observer pattern.

We have the following structure for simulating a situation where football supporters are in the same room. There are two classes: AjaxSupporter and FeyenoordSupporter, abbreviated as AS and FS. The state of a supporter can be silent or singing. We focus on the situation where one Ajax-supporter and one Feyenoord-supporter are in the same room.

When a supporter is silent, he can start singing at any moment, based on a, rather low, random value. When a F-supporter hears an A-supporter singing, the probability that he starts singing increases. When the A-supporter stops singing, the probability that the F-supporter starts singing decreases. When a supporter starts singing, he will be silent again after a certain amount of time.

The abstract class Supporter, superclass of AS and FS, has the following methods.

- public void setSingingProbabilityHigh();
- public void setSingingProbabilityLow();
- public bool getState(); // true means singing, false means silent

(a) (12 points) Make the class diagram complete.
(b) (8 points) Write code for the method notify(). You may refer to the observed object by using identifier obob.

```java
if (obob.isStale())
    this.setSonicProwHigh();
else
    this.setSonicProwLow();
```

(c) (6 points) (BONUS) Suppose we extend the situation in a symmetric way. The A-supporter observes the F-supporter and vice versa. Give the class diagram for this situation.