1. [15%] On a highway, speeding Vehicles are ticketed in a RadarTrap.
Here is one way for Vehicle v to get a speeding ticket:

\[
\begin{align*}
    &\text{v.setSpeed}(100); \quad \text{// drive faster than the speed limit,} \\
    &\text{if } (\text{v.getSpeed()} > \text{SPEED\_LIMIT}) \quad \text{// and if your speed is checked,} \\
    &\quad \text{v.getTicketed();} \quad \text{// you get ticketed.}
\end{align*}
\]

RadarTrap keeps a list of the Vehicles currently traveling through it, which are then periodically checked for speeding, as follows:

\[
\begin{align*}
    &\text{RadarTrap } r = \text{new RadarTrap();} \\
    &\text{r.enter(v);} \quad \quad \text{// v is one more Vehicle in the RadarTrap } r \\
    &\text{r.checkForSpeeders();} \quad \quad \text{// all Vehicles currently in } r \text{ are checked}
\end{align*}
\]

In class RadarTrap, below, the List of vehicles has method iterator() which returns an Iterator (which has methods hasNext() and next()). Finish writing the method checkForSpeeders() which iterates through the vehicles, and any Vehicle v found speeding by measureSpeed(v) will v.getTicketed().

```java
public class RadarTrap {
    private int SPEED\_LIMIT = 65;
    private List<Vehicle> vehicles = new SinglyLinkedList<Vehicle>();

    public void enter(Vehicle v) { vehicles.add(v); }

    public int measureSpeed(Vehicle v) {
        v.illuminated(this); \quad \text{// v is first illuminated by the RadarTrap}
        return v.getSpeed(); \quad \text{// and then its’s measured speed is returned}
    }

    public void checkForSpeeders() {
```
There are three patterns that you should apply in the following: **Decorator, Strategy** and **Observer**. First, the Decorator Pattern will be used in combination with the Strategy Pattern in question 2.

Notice that the RadarTrap must first *illuminate* the vehicle with radar, by calling `v.illuminated(this)`, passing itself as the argument:

```java
public int measureSpeed(Vehicle v) {
    v.illuminated(this);   // v is illuminated by the RadarTrap
    return v.getSpeed();  // and then return its speed
}
```

2) [25%] First, regarding the **Decorator Pattern**, any Vehicle can be decorated with a radar detector (call this kind of Vehicle **RadarDetecting**). This will permits a different *strategy* when it detects it is being illuminate by radar.

```java
Vehicle v = new Ferrari();  // start off with some car or another
v = new RadarDetecting(v);  // and decorate it with a radar Detector
```

Second, regarding the **Strategy Pattern**, there are two strategies to consider for its *illuminated* method: 1) do nothing (just keep on driving unaware), or 2) immediately slow down. The Decorator Pattern allows the second strategy.

Use a **UML class diagram** to show a design incorporating both the Decorator and Strategy Patterns. Then (assuming RadarTrap `r` and Vehicle `v`) provide a **UML sequence diagram** for the method call `v.illuminated(this)` in the above measureSpeed method.

3) [10%] The Decorator Pattern was proposed rather than just the subclass RadarDetecting. Why? What is the advantage of the Decorator Pattern specifically over simply making that a subclass?

4) [25%] Regarding the **Observer Pattern**, three applications come to mind. One is that the Vehicles are Observable and the RadarTrap is the Observer. The second is that a RadarDetecting, once illuminated, can begin to observe the RadarTrap (which was the argument in illuminate, remember). The third is that both are Observers and Observables. Choose one or the other (or both) and using UML (class and sequence), and any code snippets, explain your design (including what changes are needed to RadarTrap and the Vehicle hierarchy.

5) [25%] Provide code snippets for your version of the Strategy Pattern and the Decorator Pattern for either question 2 or question 3. Write the Strategy hierarchy, plus Vehicle, Decorator, and RadarDetecting.