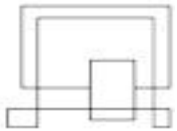
Life is great
VIA University College



Software Development with UML and Java 2

Today’s plan

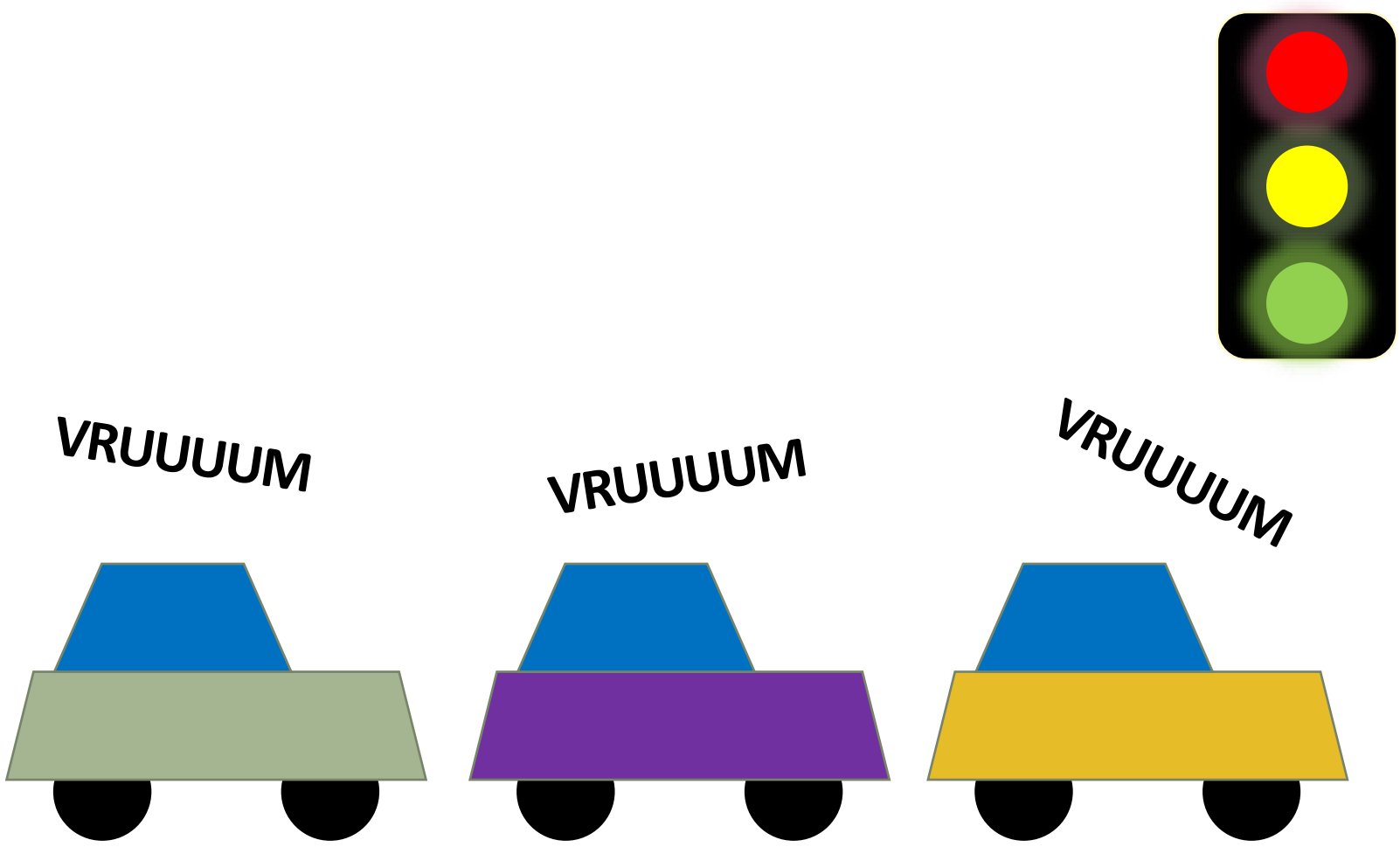
- Trying a new approach

- First an exercise, 20 min
- Talk about exercise
- Then an exercise, 20 min
- Talk about exercise
- Then today’s topic
- Then informative stuff
- Then more exercises

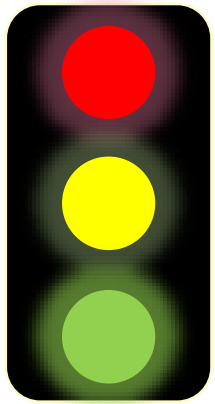
The exercise

- About cars reacting to the change of lights of a traffic light

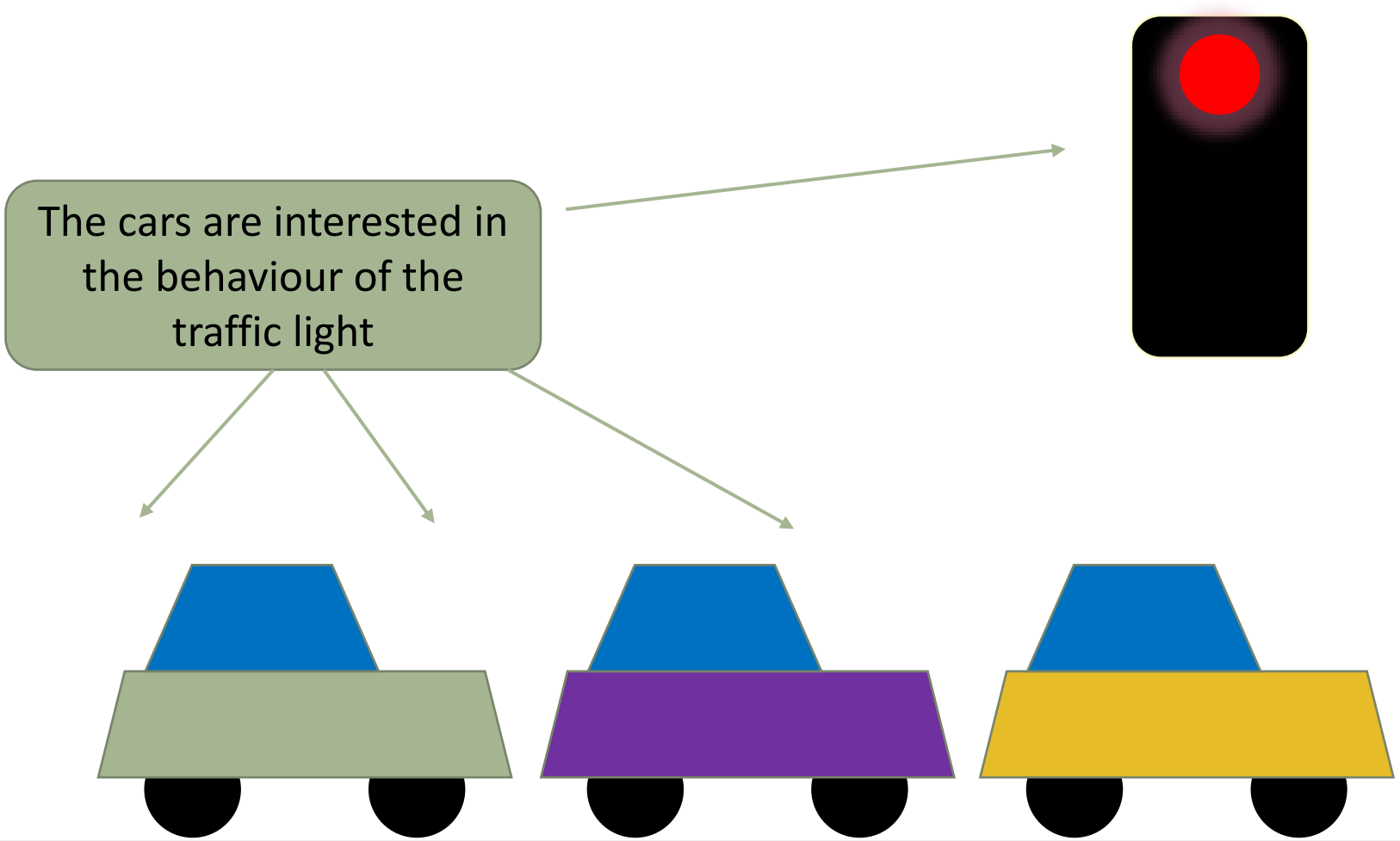
Traffic light example



Traffic light example



Traffic light example



Implementation, your exercise

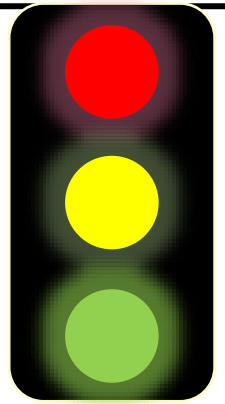
- Traffic light class
- Car class (multiple instances)

- Light changes → Cars react.

- I’ll give you parts of the code

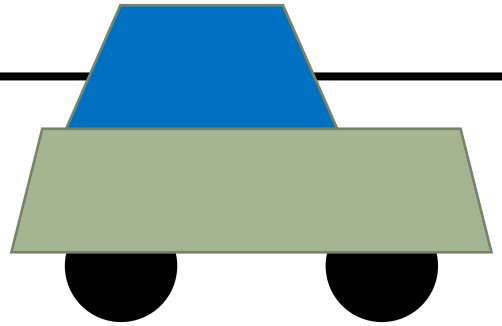
- We need a main method to instantiate the traffic light and cars

```
    private String[] lights = {"GREEN",
                    "YELLOW", "RED", "YELLOW"};

     private int lightIndex = 2;

     public void SimulateTrafficLight()
throws
    InterruptedException {
        for (int i = 0; i < 10; i++) {
            Thread.sleep(1000);
            lightIndex = (++ lightIndex) % 4;
            currentLight = lights[lightIndex];
            System.out.println("\nLight is "
                            + currentLight);
        }
     }
```

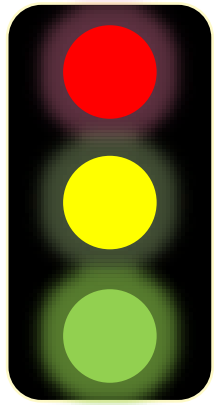


```
public void ReactToLight(String currentLight)
{
    if("GREEN".equals(currentLight)) {
        System.out.println("Car " +
                           id + " drives");
    } else if("YELLOW".equals(currentLight)) {
        if("RED".equals(previousLight)) {
            System.out.println("Car " + id +
                       " turns engine on");
        } else {
            System.out.println("Car " + id + "
                           slows down");
        }
    } else if("RED".equals(currentLight)) {
        System.out.println("Car " + id
                         + " stops");
    }
    previousLight = currentLight;
}
```

Notice field variables marked with purple text



```
private String[] lights = {"GREEN",
                "YELLOW", "RED", "YELLOW"};

private int lightIndex = 2;

public void start() throws
                    InterruptedException {
    for (int i = 0; i < 10; i++) {
        Thread.sleep(1000);
        lightIndex = (++ lightIndex) % 4;
        currentLight = lights[lightIndex];
        System.out.println("\nLight is "
                        + currentLight);
    }
}
```



Follow up

- How did you solve it?

Next step

- Add two new traffic light watchers:
  - Pedestrian -> stops for green, walks when the light is red
  - Taxi -> doesn’t care about yellow

```
public void setLight(String currentLight) {
    if ("GREEN".equals(currentLight)) {
        System.out.println("Taxi " + id + " drives super fast");
    } else if ("RED".equals(currentLight)) {
        System.out.println("Taxi " + id + " stops with screething tires");
    }
}
```

```
public void setLight(String currentLight) {
    if ("GREEN".equals(currentLight)) {
        System.out.println("Pedestrian " + id + " waits");
    } else if ("RED".equals(currentLight)) {
        System.out.println("Pedestrian " + id + " walks");
    }
}
```

Follow up

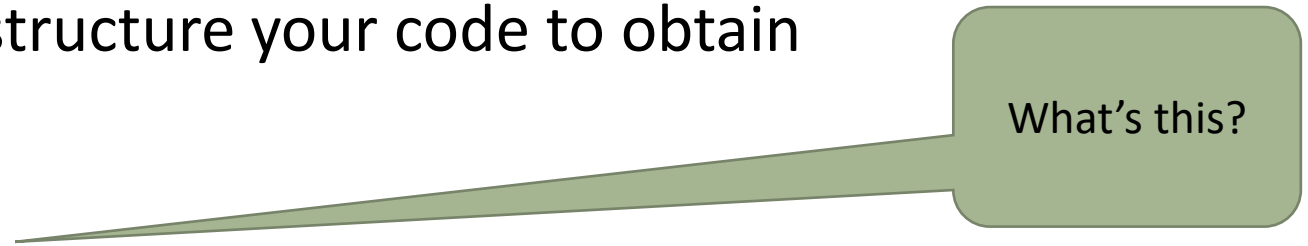
- How did you solve it?

Agenda

- What is a design pattern?
- What is the Observer design pattern?
  - Concept
  - Definition of purpose
- What does the Observer design pattern look like?
  - UML Structure
- Why even use the Observer design pattern?

Design patterns in general

- **General solutions to common problems**
- Usually a way to structure your code to obtain
  - Flexibility
  - Reusability
  - Loose couplings
  - Improved code structure/architecture



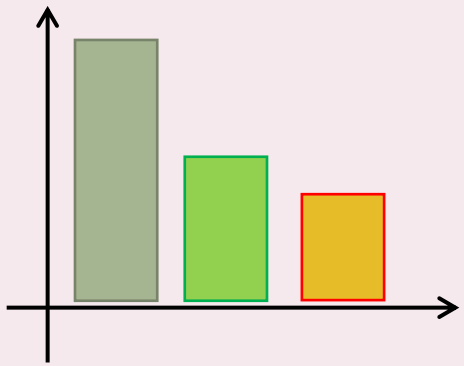
- Almost every pattern uses interfaces one way or the other
- Each pattern is used to solve a specific problem
- We'll see about 8 patterns in SDJ2
- There are 24 famous "basic" patterns, from Gamma et al. (1995)
- Plenty have appeared since

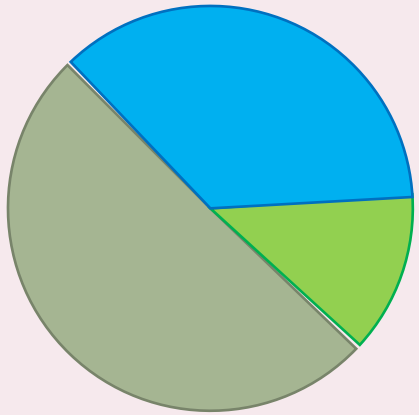
Agenda

- What is a design pattern?
- What is the Observer design pattern?
  - Concept
  - Definition of purpose
- What does the Observer design pattern look like?
  - UML Structure
- Why even use the Observer design pattern?

Listeners

	A	B	C
X	60	25	15
Y	50	30	20
Z	40	35	25

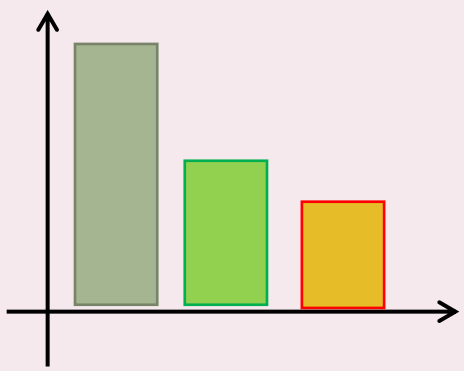


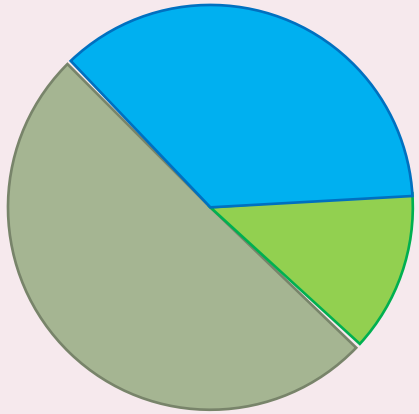


A = 50%
B = 30%
C = 20%

Subject

	A	B	C
X	60	25	15
Y	50	30	20
Z	40	35	25



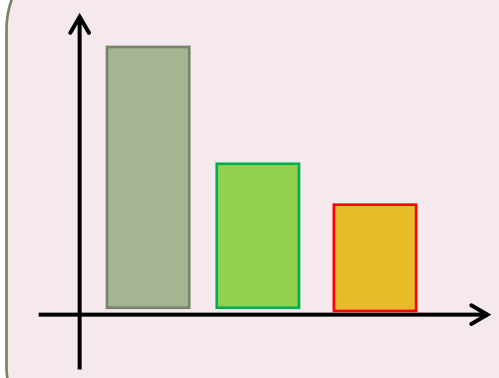


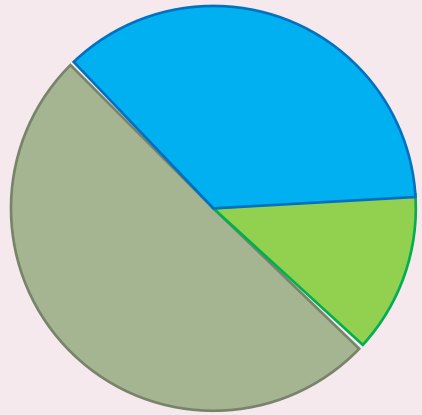
A = 60%

B = 10%

C = 30%

	A	B	C
X	60	25	15
Y	50	30	20
Z	40	35	25



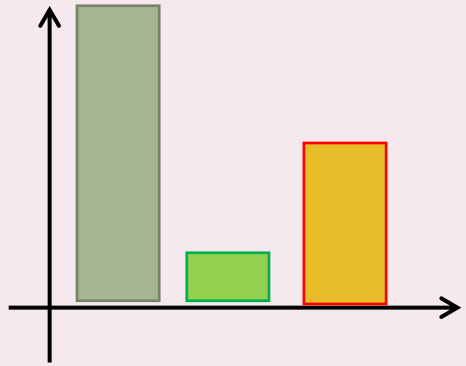


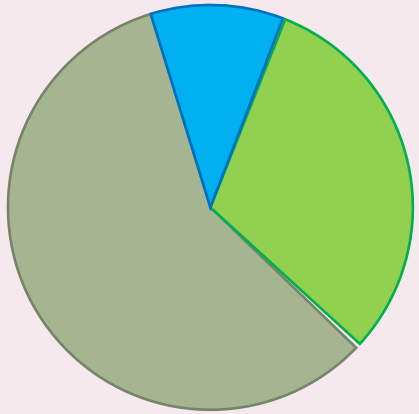
A = 60%

B = 10%

C = 30%

	A	B	C
X	60	25	15
Y	60	10	30
Z	40	35	25





A = 60%

B = 10%

C = 30%

Soccer example

- Online betting sites
- Online live-score
- Medics
- SMS live updates
- …

Observers observing an observable

Subject

Listeners

Observers observing an observable

Wauw!

Did you hear that?

Amazing voice

- In all cases many objects are interested in one object
- ***Listeners*** are interested in the ***Subject***

Agenda

- What is a design pattern?
- What is the Observer design pattern?
  - Concept
  - Definition of purpose
- What does the Observer design pattern look like?
  - UML Structure
- Why even use the Observer design pattern?

Design Pattern: Observer

- Problem
  - You have a number of (maybe different) classes, which are interested in the state/data/behaviour of another class.
- Intent
  - Define a many-to-one relationship between objects so that when one object changes state, all its interested objects are notified
  - One broadcast to many
- Example
  - One class calculates some data, and other classes want to display them in different ways
  - Button listeners, listeners in JavaFX

Observer design pattern – the basic idea

1) A class (Listener) subscribes to a service (Subject)
2) Listener gets a “message” every time there is an update
3) Listeners act upon the update

Agenda

- What is a design pattern?
- What is the Observer design pattern?
  - Concept
  - Definition of purpose
- What does the Observer design pattern look like?
  - UML Structure
- Why even use the Observer design pattern?
- Example

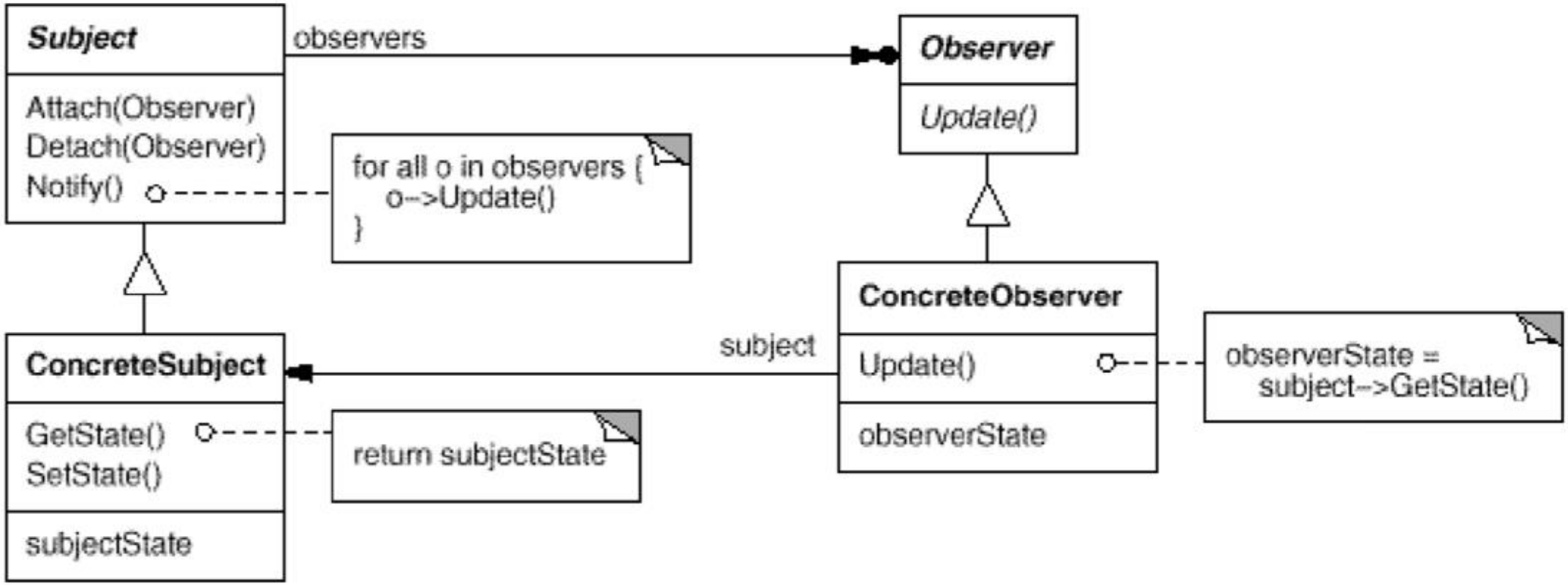
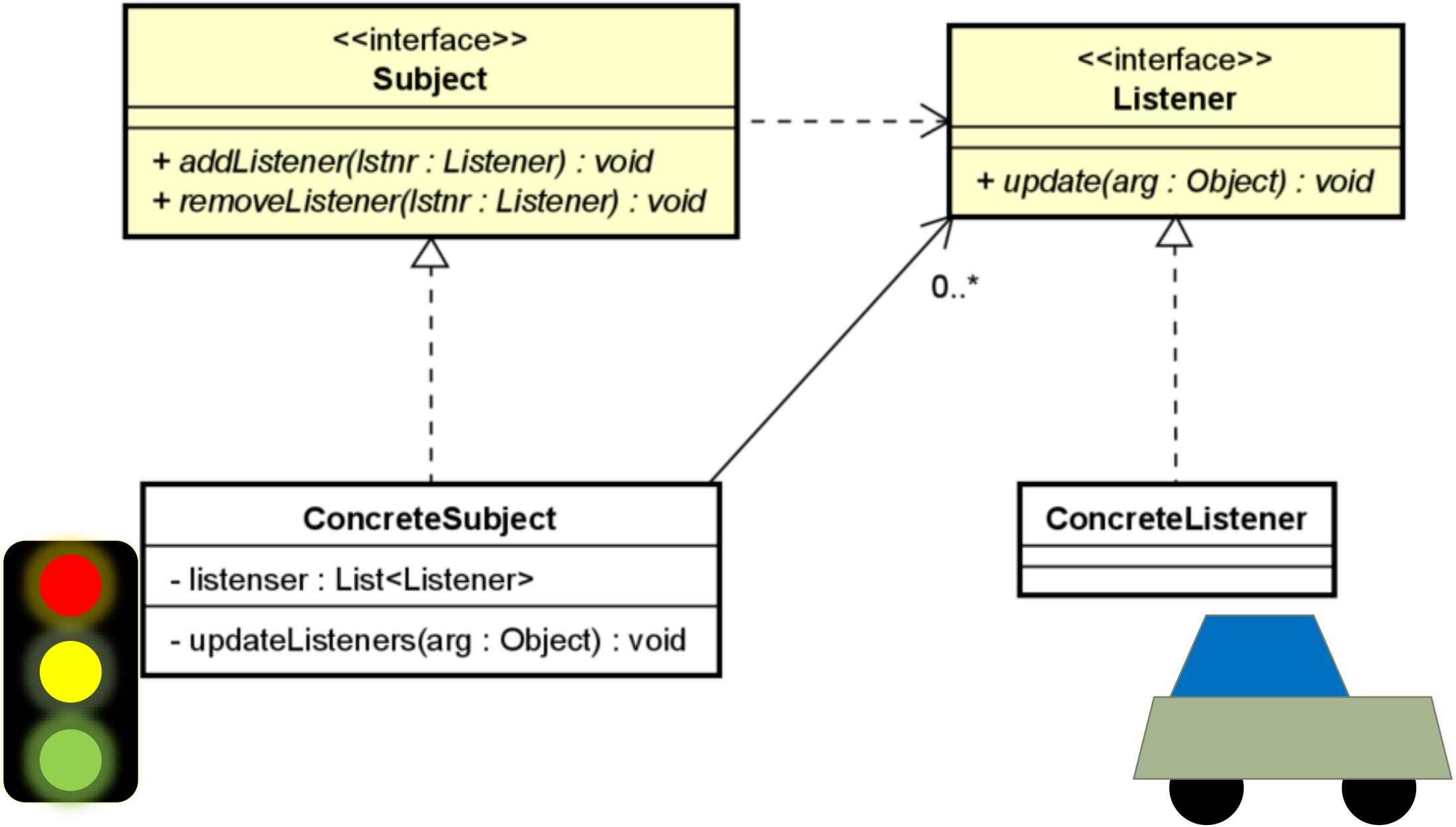
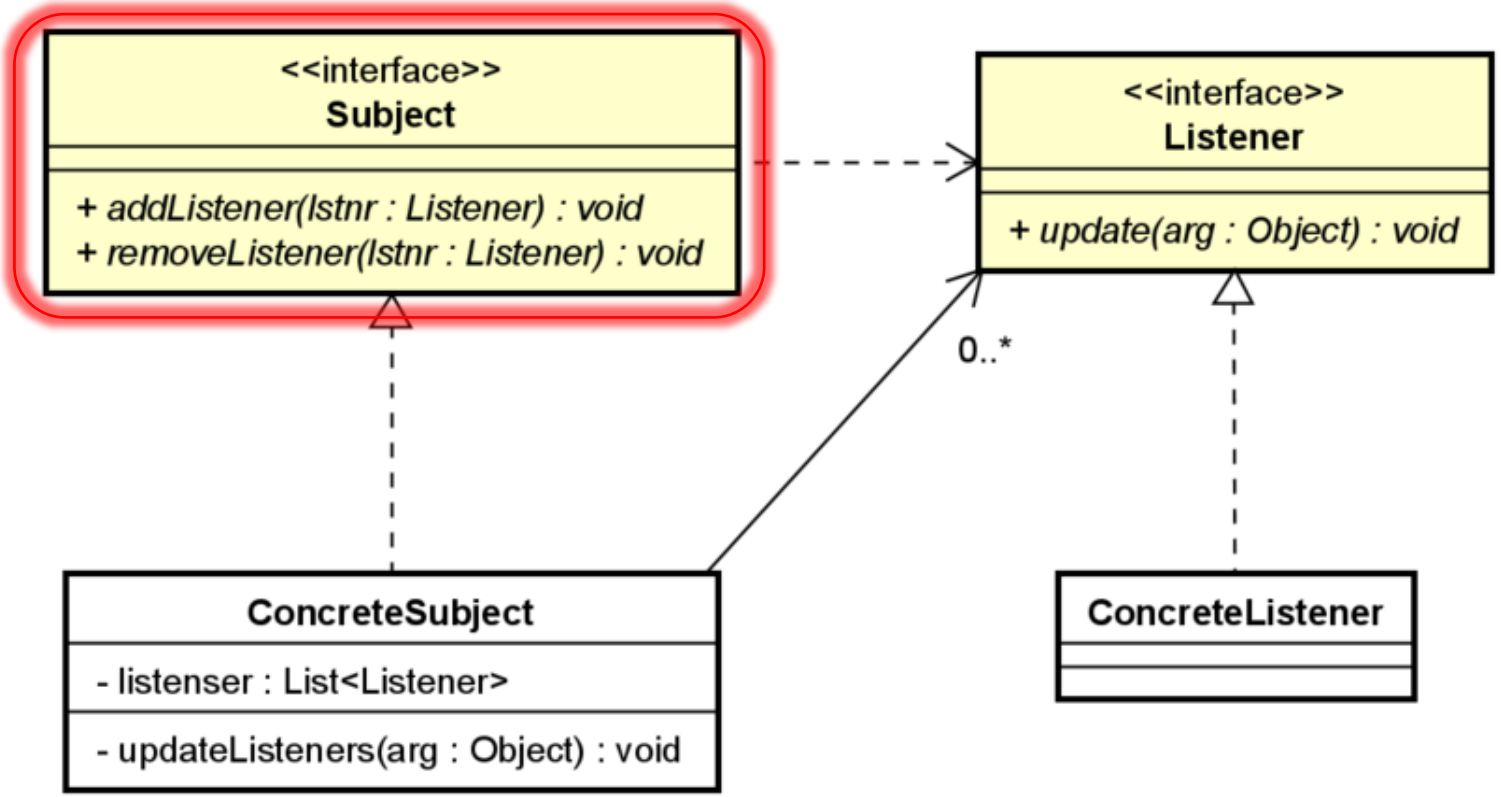
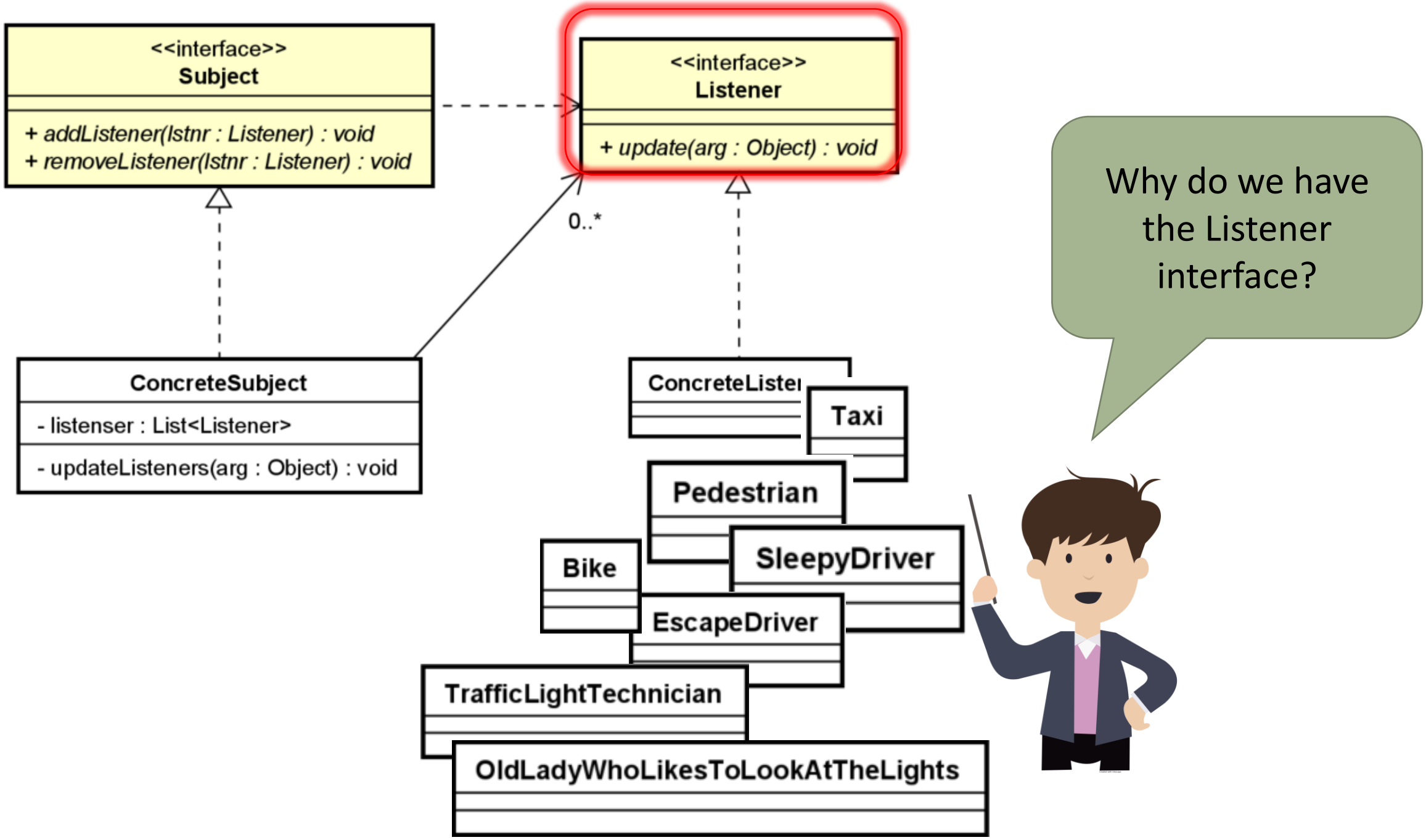


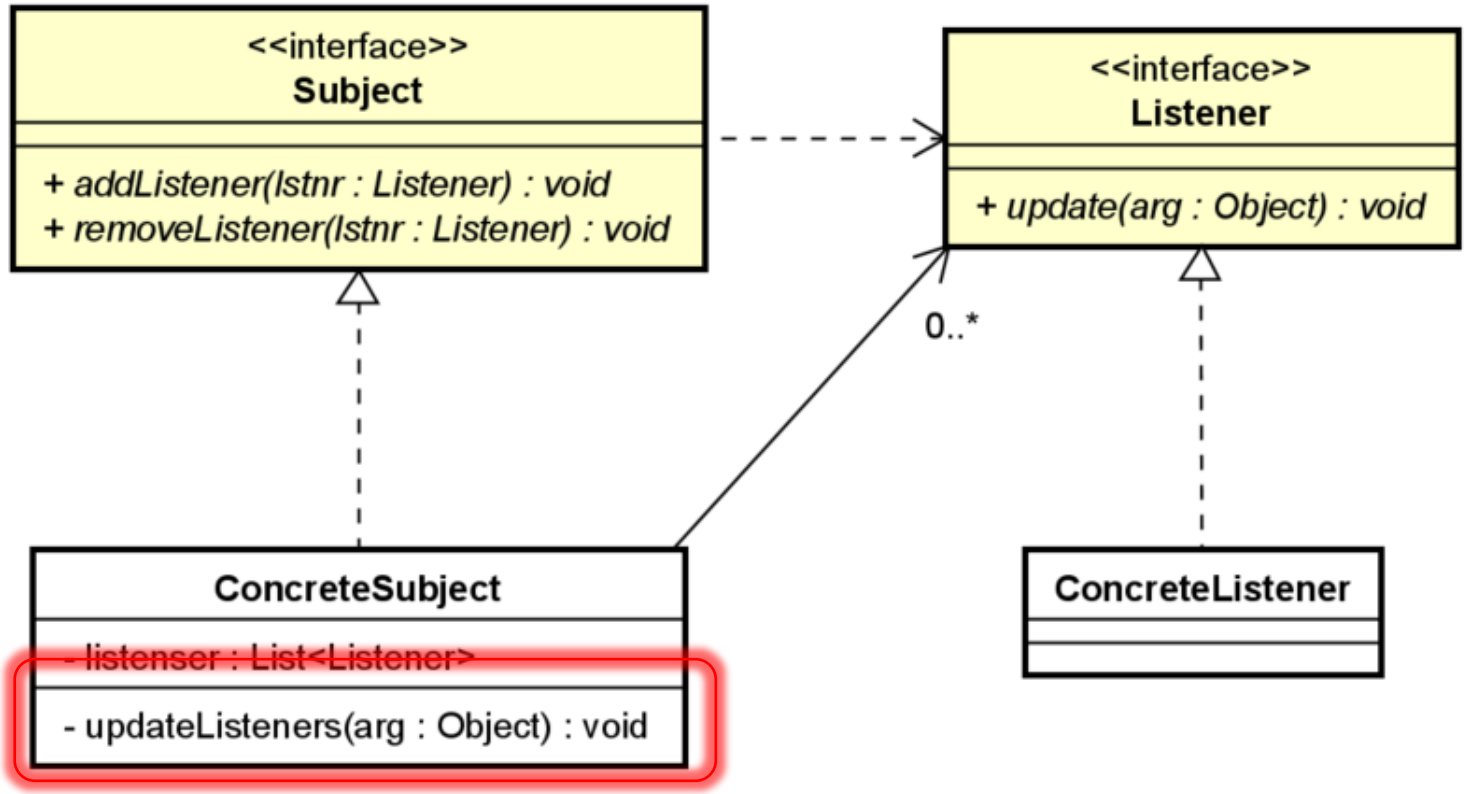
Figure 1: Observer (Gamma, et al., 1995) page 294.


<<interface>>
Subject
+ addListener(lstnr : Listener) : void
+ removeListener(lstnr : Listener) : void
<<interface>>
Listener
+ update(arg : Object) : void
0..*
ConcreteSubject
- listenser : List<Listener>
- updateListeners(arg : Object) : void
ConcreteListener



Why do we have the Subject interface?


<<interface>>
Subject
+ addListener(lstnr : Listener) : void
+ removeListener(lstnr : Listener) : void
<<interface>>
Listener
+ update(arg : Object) : void
0..*
ConcreteSubject
- listenser : List<Listener>
- updateListeners(arg : Object) : void
ConcreteListe
Taxi
Pedestrian
Bike
SleepyDriver
EscapeDriver
TrafficLightTechnician
OldLadyWhoLikesToLookAtTheLights
Why do we have the Listener interface?

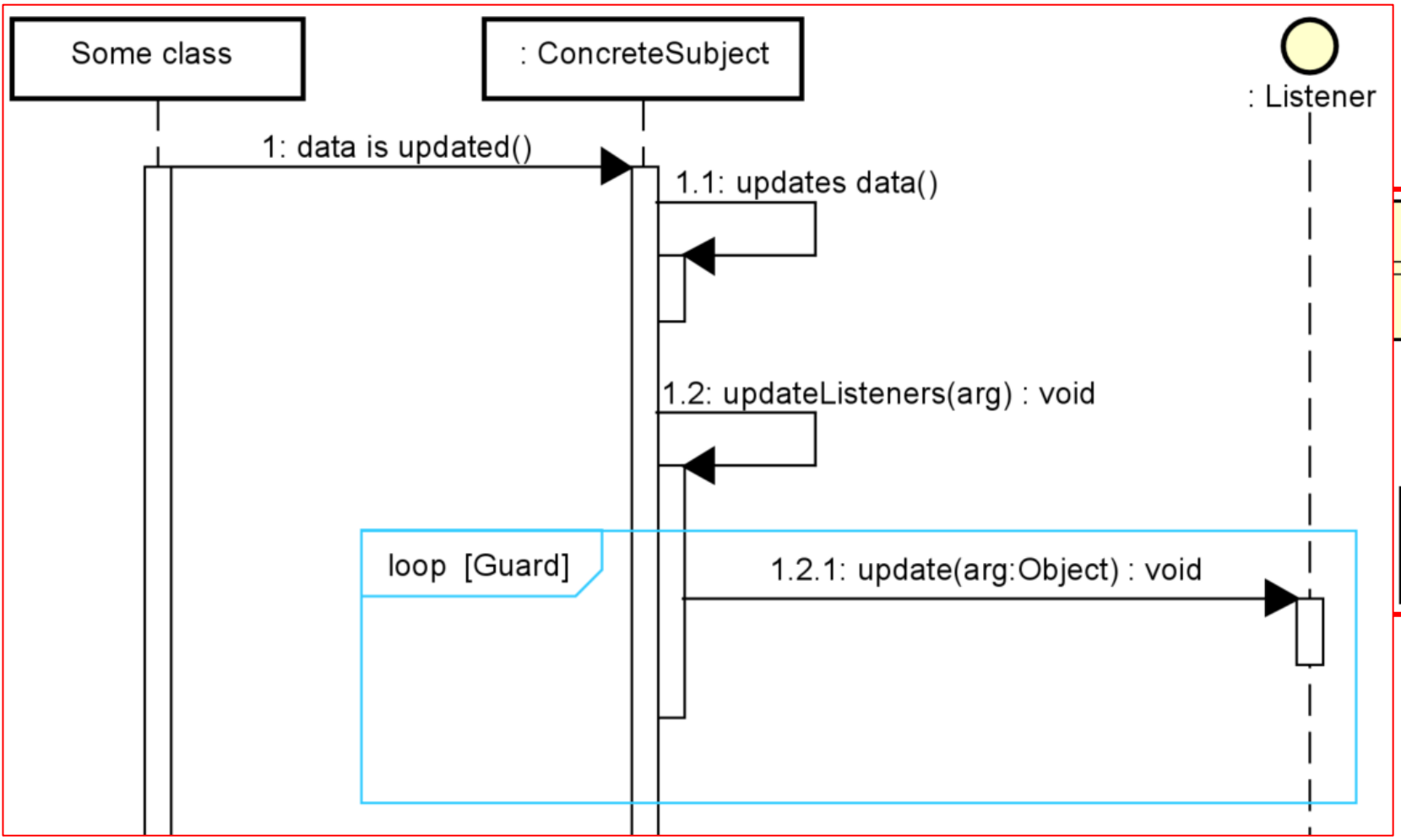


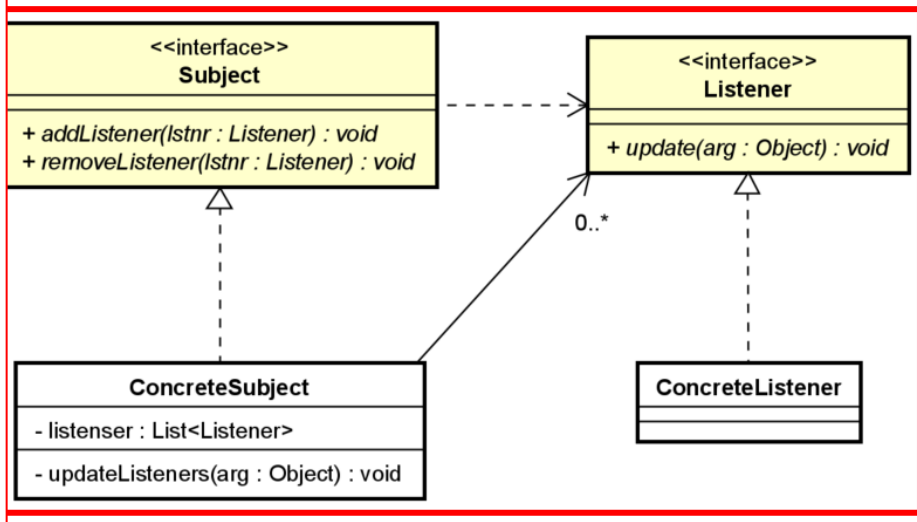

```
private void updateListeners(Object arg) {
    for (Listener listener : listeners) {
        listener.update(arg);
    }
}
```

What do we put in updateListeners method?

We say “the subject fires an event”

Have you seen this type of loop?






```
private void updateListeners(Object arg) {
    for (Listener listener : listeners) {
        listener.update(arg);
    }
}
```

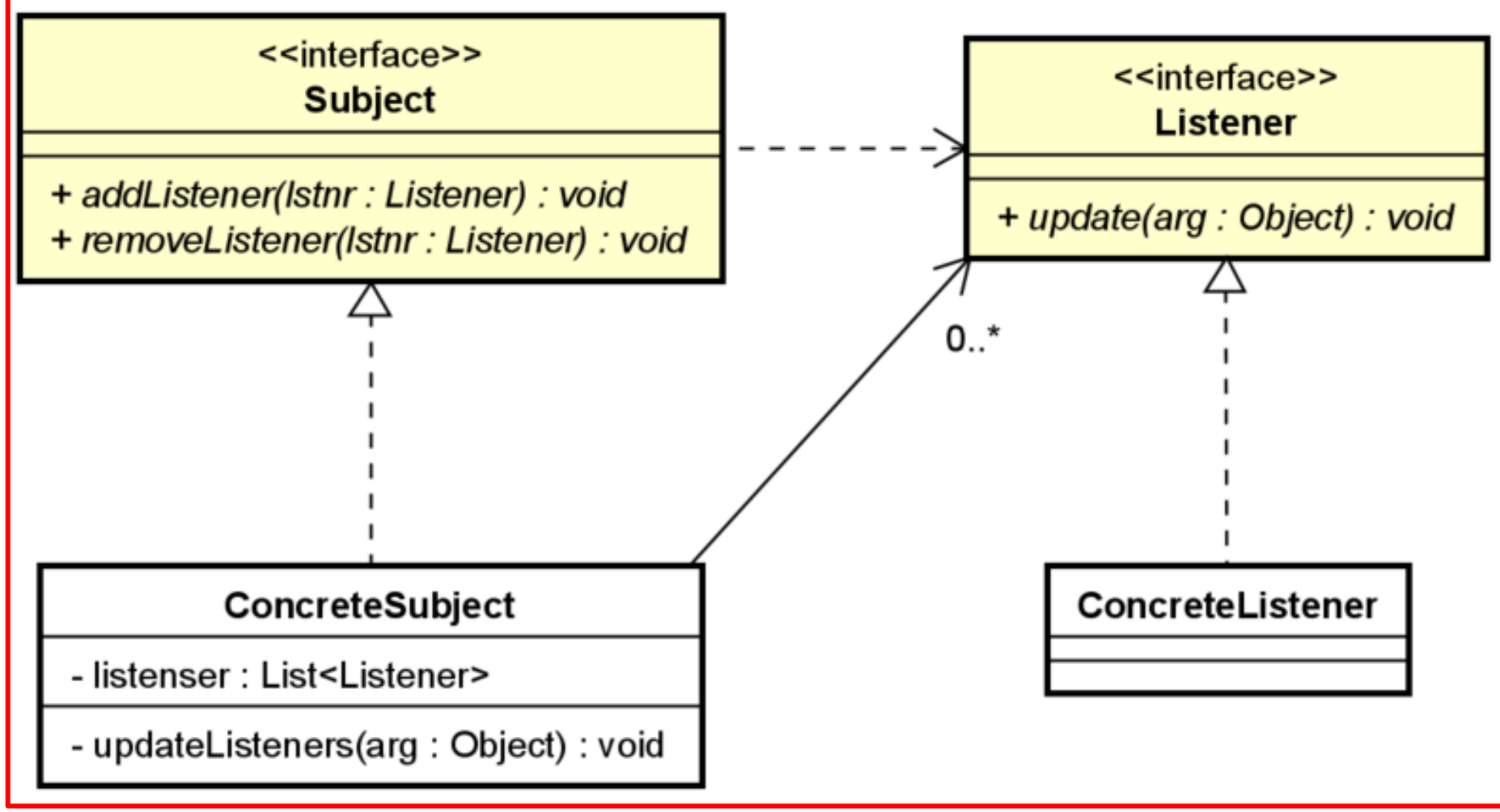
Agenda

- What is a design pattern?
- What is the Observer design pattern?
  - Concept
  - Definition of purpose
- What does the Observer design pattern look like?
  - UML Structure
- Why even use the Observer design pattern?
- Example

Pros/cons?

- Benefits?

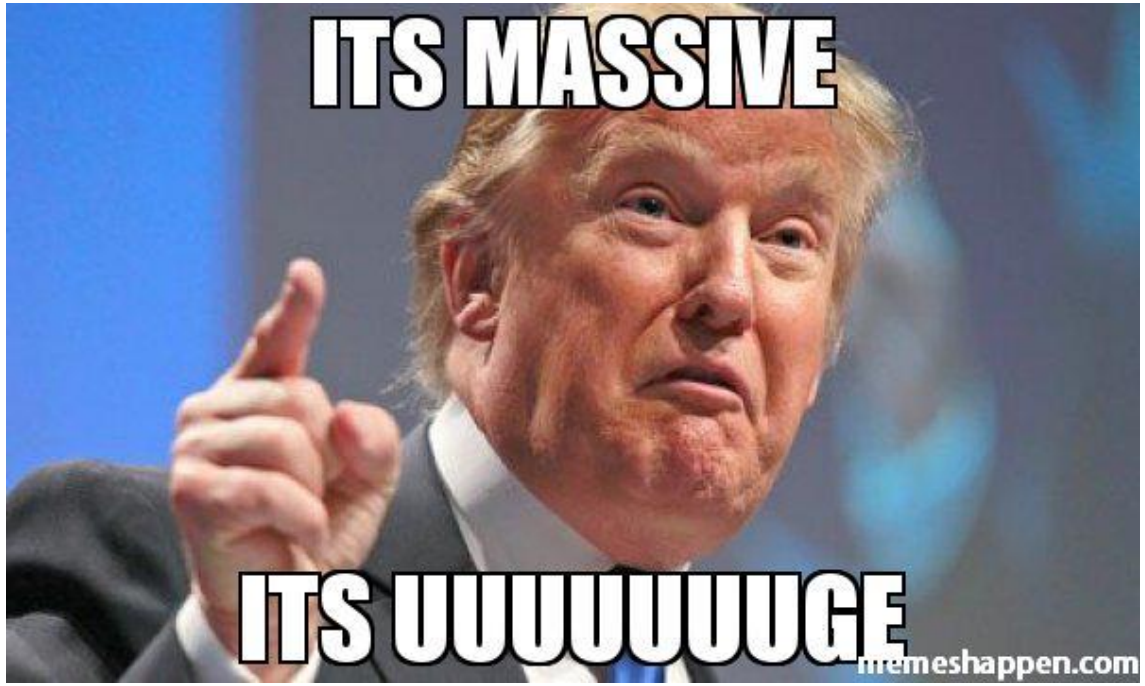
- Drawback?



- The traffic light doesn't care that I added a new Listener, the Taxi.
- It will just notify all who's interested that something changed
- I can now add all kinds of new Listeners, e.g. escape drivers, pedestrians, etc.

without changing existing code.

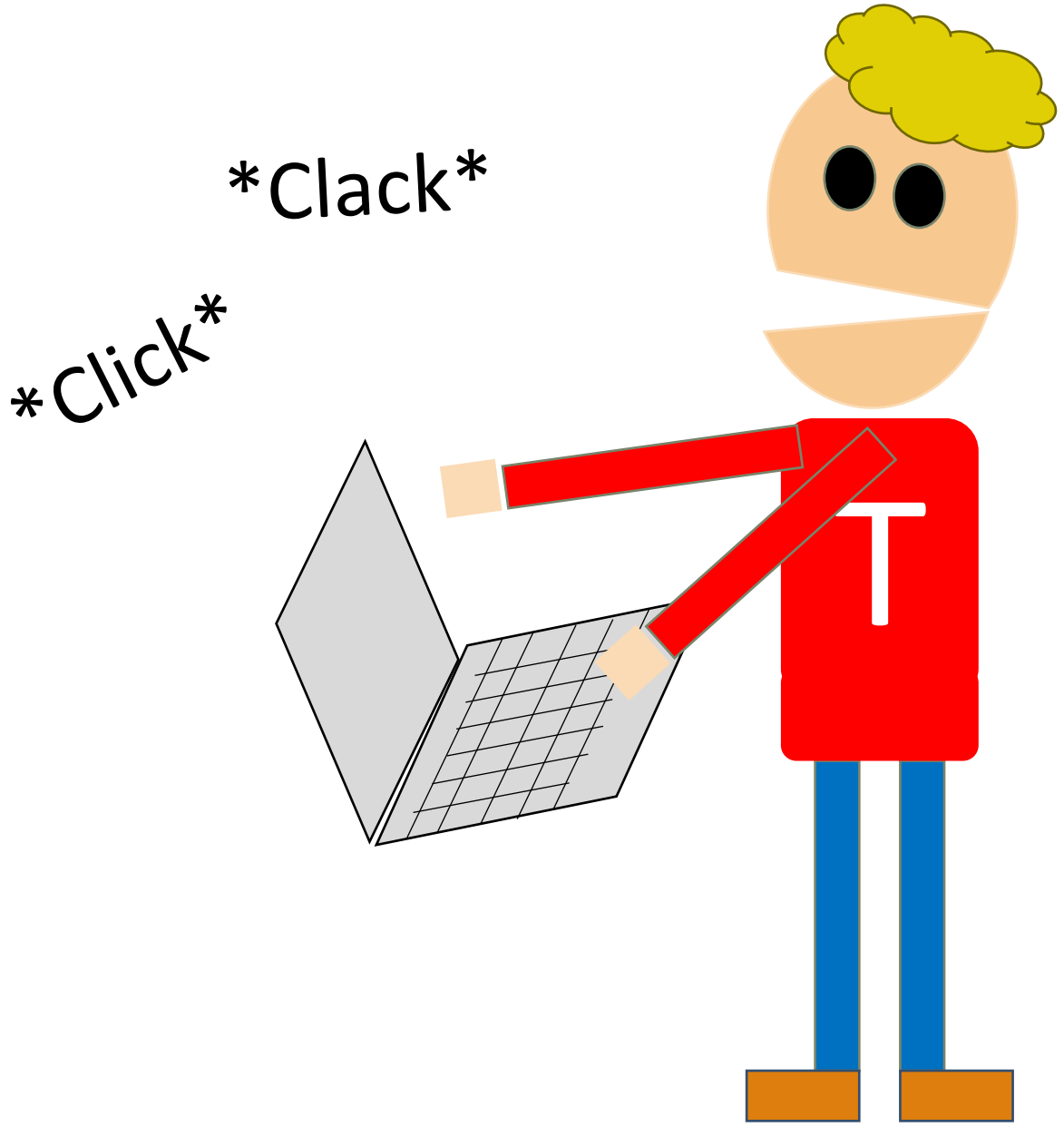
Huge benefit.
Loose coupling



Agenda

- What is a design pattern?
- What is the Observer design pattern?
  - Concept
  - Definition of purpose
- What does the Observer design pattern look like?
  - UML Structure
- Why even use the Observer design pattern?
- Example

Applying the observer pattern to the traffic light example



Agenda

- What is a design pattern?
- What is the Observer design pattern?
  - Concept
  - Definition of purpose
- What does the Observer design pattern look like?
  - UML Structure
- Why even use the Observer design pattern?
- Example

Informative stuff

- Two videos to follow up on the Observer pattern.
- Debugging tutorial
- Notice comments below slides.

- Exercises

- Britney Spears