Software Design

- Concepts
  - Abstraction & encapsulation (Feb 1)
  - Design patterns
    - Observer (Today)
    - State, Strategy
  - Dependency inversion
  - Dependency injection

Software Design

- Learning objectives for today:
  - Given a design/code and an appropriate feature request:
    - apply the Observer design pattern to implement the feature
    - explain the benefits and limitations of the use of the Observer design pattern for the design/code and feature request

Plan for Today

Joint design a new feature for the Mario game

Brief overview of Design Patterns

Observer Pattern

Apply Observer Pattern for the feature in Mario

- Sketch design
- Code

Exercise in applying Observer

New Feature for Mario

When a Gumpa appears,
    have Mario “do something”

When a Gumpa dies,
    have Mario “do something else”

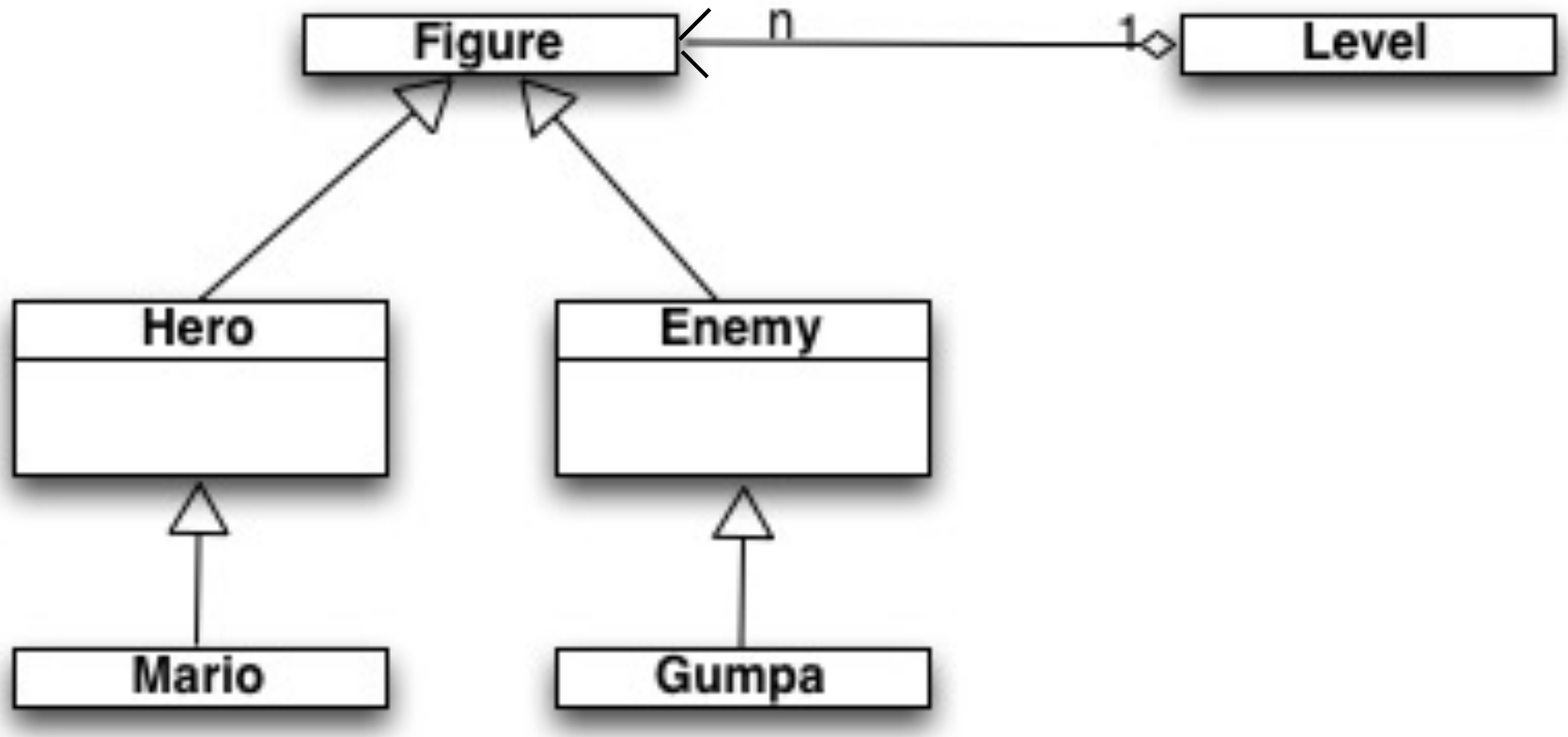


Figure
n
1
Level
Hero
Enemy
Mario
Gumpa

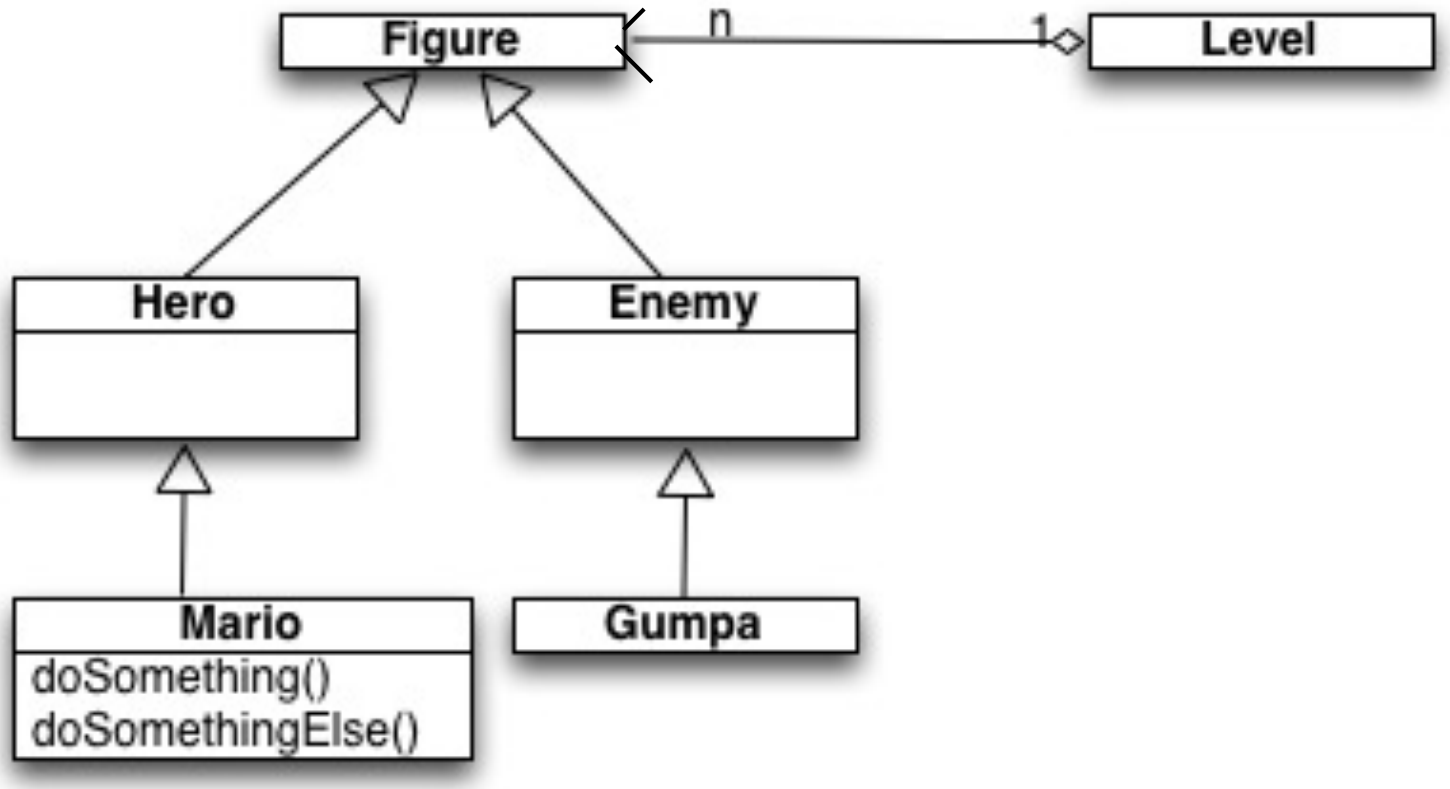
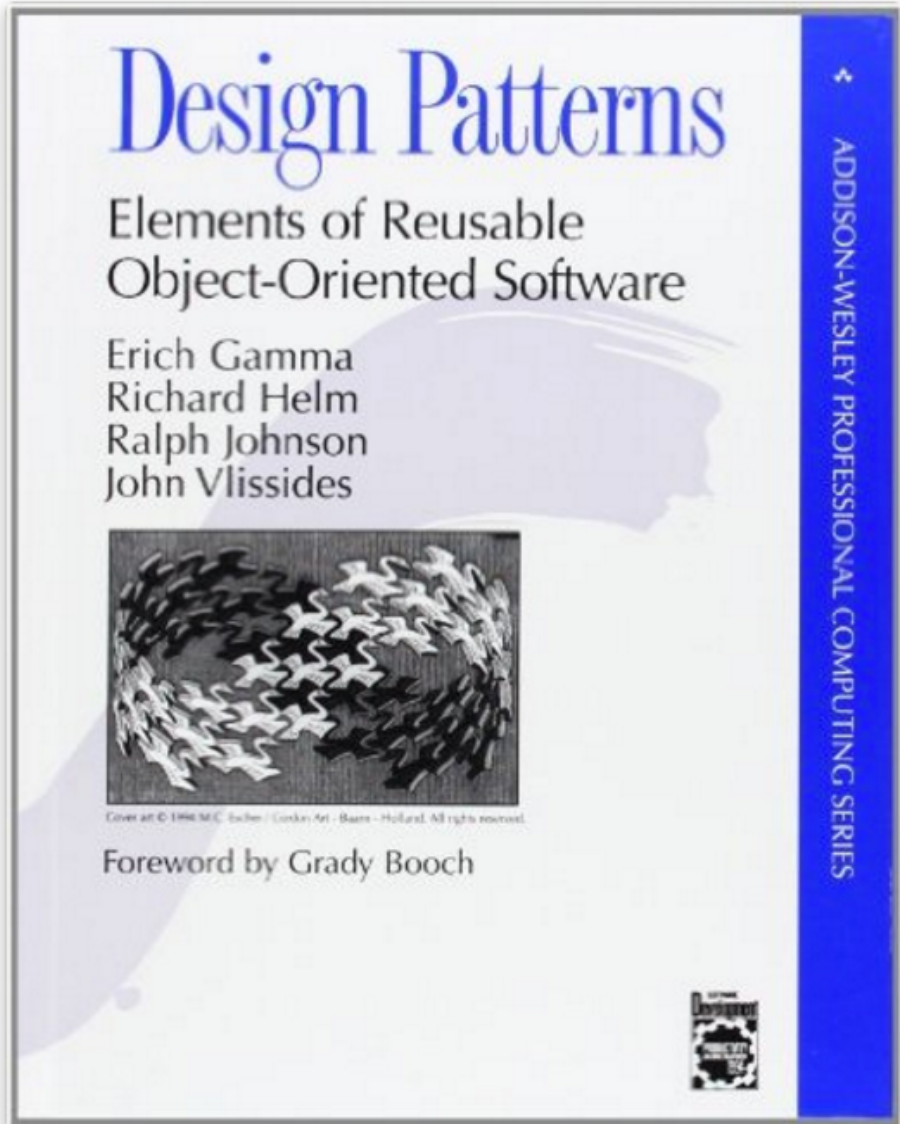


Figure
n
1
Level
Hero
Enemy
Mario
doSomething()
doSomethingElse()
Gumpa

How do we call the Mario methods?

- Think through what it means to:
  - pass Mario into the Gumpa constructor and save a reference to Mario
    - is it possible to pass Mario into Gumpa's constructor?
    - do more classes what knowledge of each other in the design? (i.e., same or more coupling?)
  - have methods in the Level class call the right methods on the right events
    - what happens to the comprehensibility, maintainability of Level?

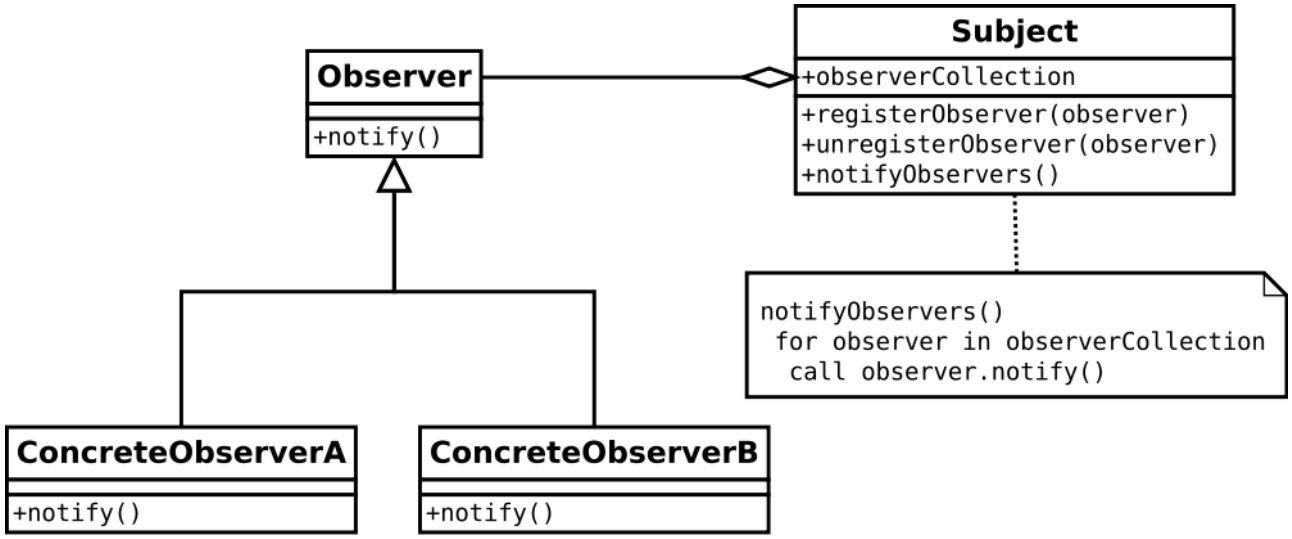
Design Patterns



Observer Design Pattern

“The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.”

— Wikipedia entry



Apply Observer Design Pattern to “New Feature for Mario”

Some steps

- What are the critical roles in the Observer Design Pattern?
  - *Subject*, the class for which a state change should be observed, e.g., Gumpa
  - *Observer*, the class to be told when the state change happens so it can react, e.g., Mario
- How are we going to map the roles onto the classes in the Mario5TS game?
- How are we going to trigger the right methods to be called, e.g., add observers to subject?

How's This?

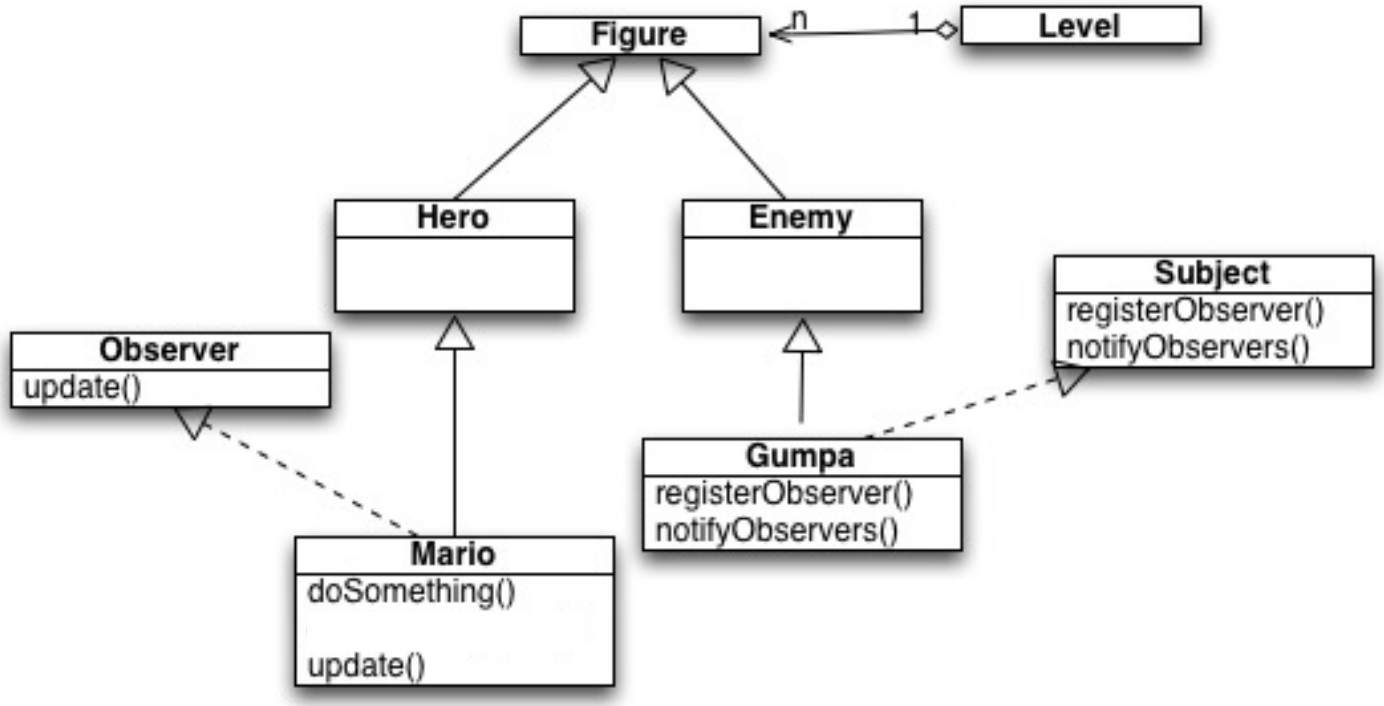


Figure
n
1
Level
Hero
Enemy
Subject
registerObserver()
notifyObservers()
Observer
update()
Gumpa
registerObserver()
notifyObservers()
Mario
doSomething()
update()

Or Maybe?

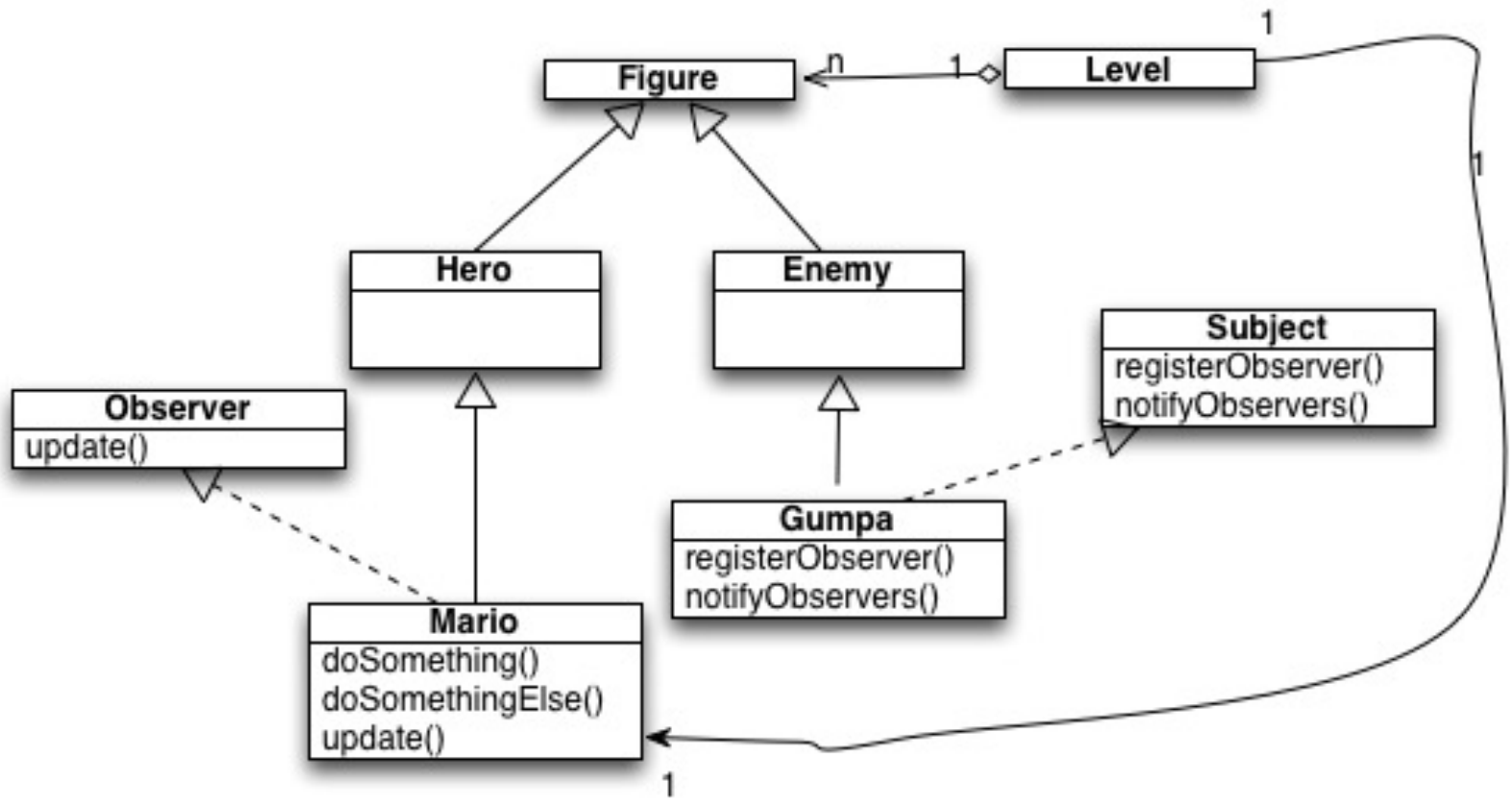


Figure
n
1
Level
1
1
Hero
Enemy
Subject
registerObserver()
notifyObservers()
Observer
update()
Gumpa
registerObserver()
notifyObservers()
Mario
doSomething()
doSomethingElse()
update()
1

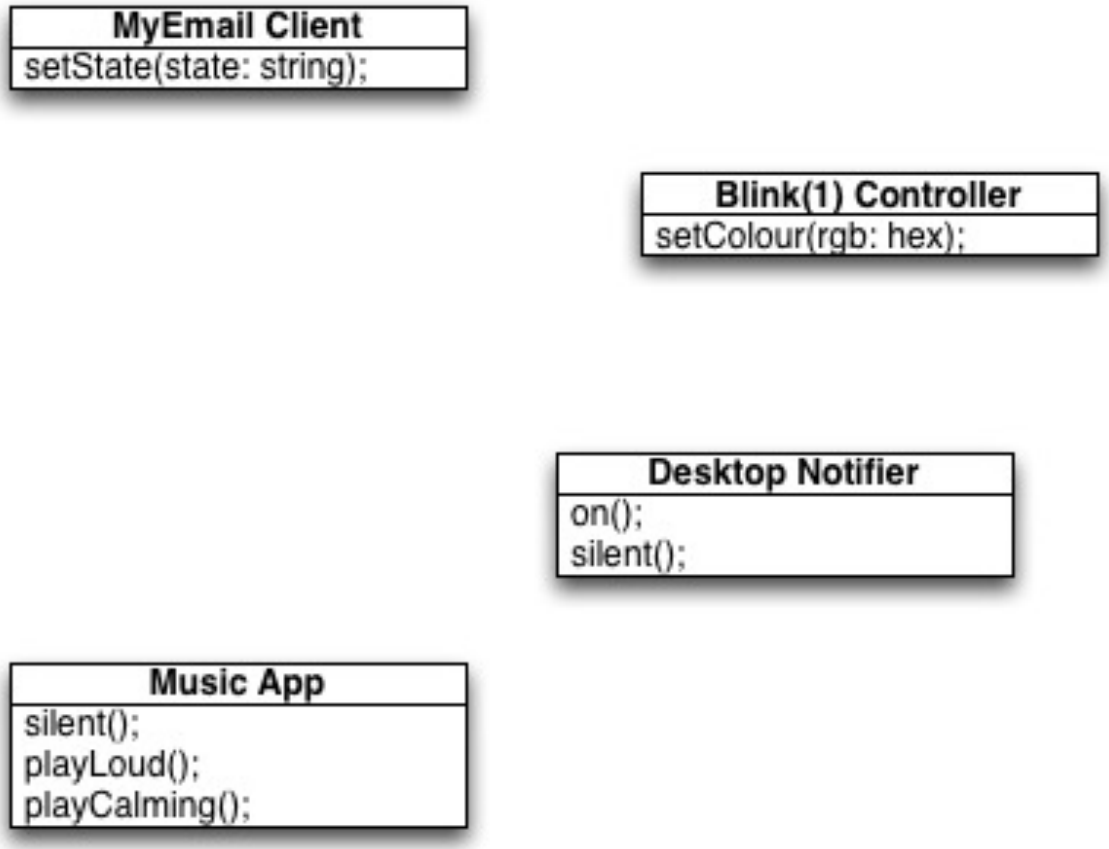
An Exercise

You are designing an email application for which you want to provide information to the user on their workflow.

The email application will recognize certain states: *overload* (more than 100 unread emails), *deluge* (in the last half hour, there has been an email every twenty seconds) and *all-is-well* (less than 5 unread emails).

When in *overload* state, a blink(1) will turn red. When in *deluge* state, desktop notifications will be silenced. When in *all-is-well* state your computer will play calming music.

You have been given the class diagram on the next slide. Apply the Observer pattern to make the desired behaviour happen.


MyEmail Client
setState(state: string);
Blink(1) Controller
setColour(rgb: hex);
Desktop Notifier
on();
silent();
Music App
silent();
playLoud();
playCalming();