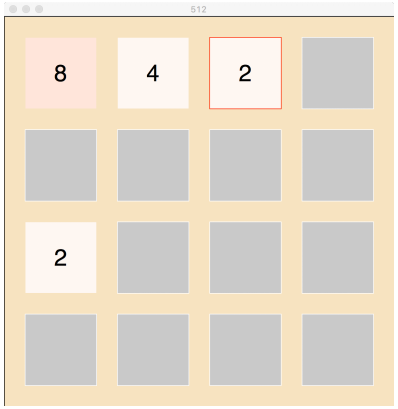
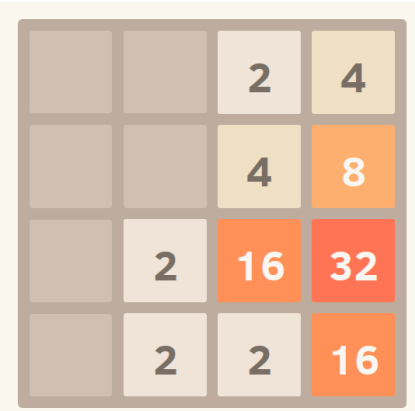
Project 3: FiveTwelve



2048 is twice 1024.

FiveTwelve is half of 1024.



Let's play

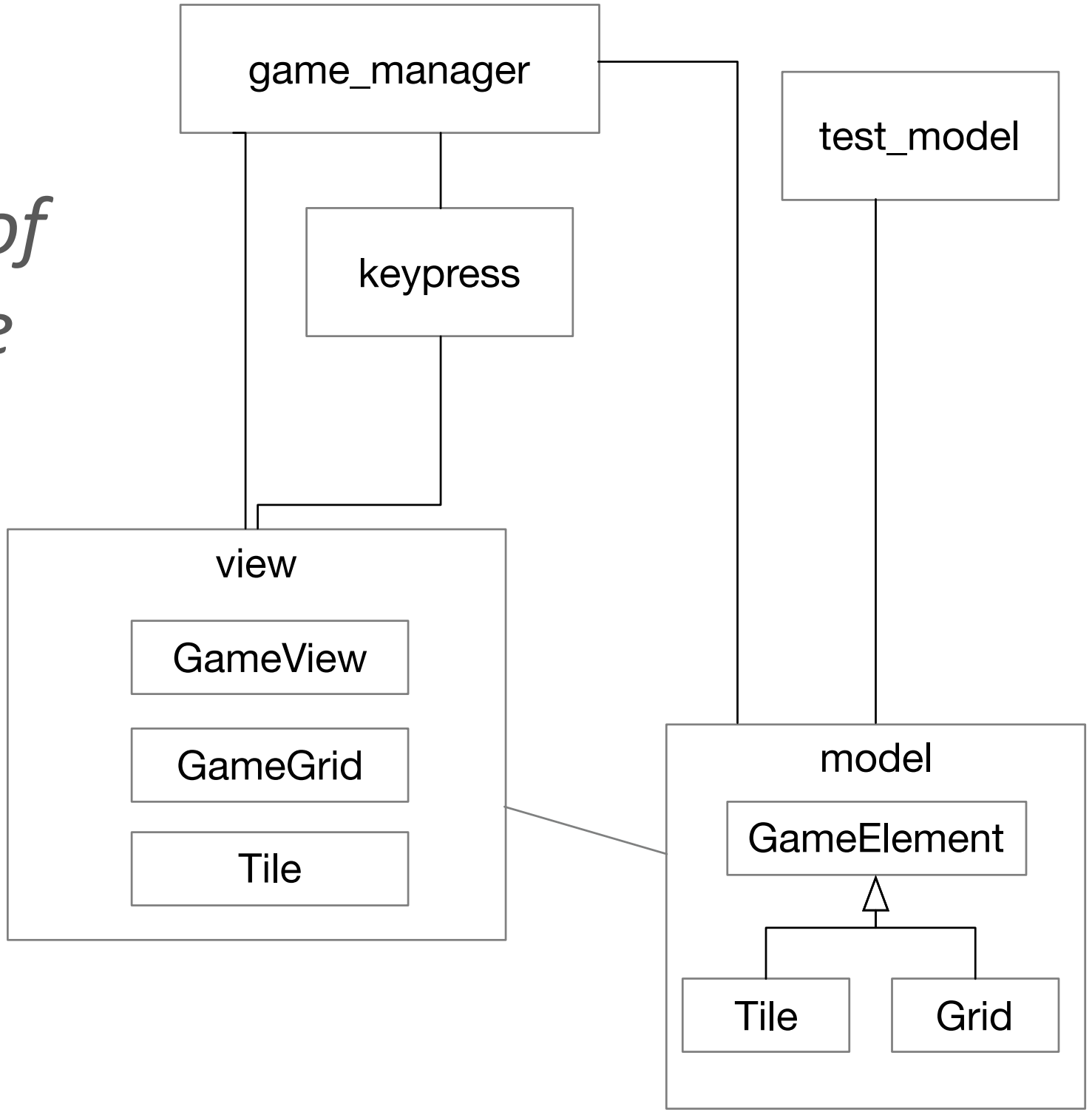
(insert demo here)

Objectives

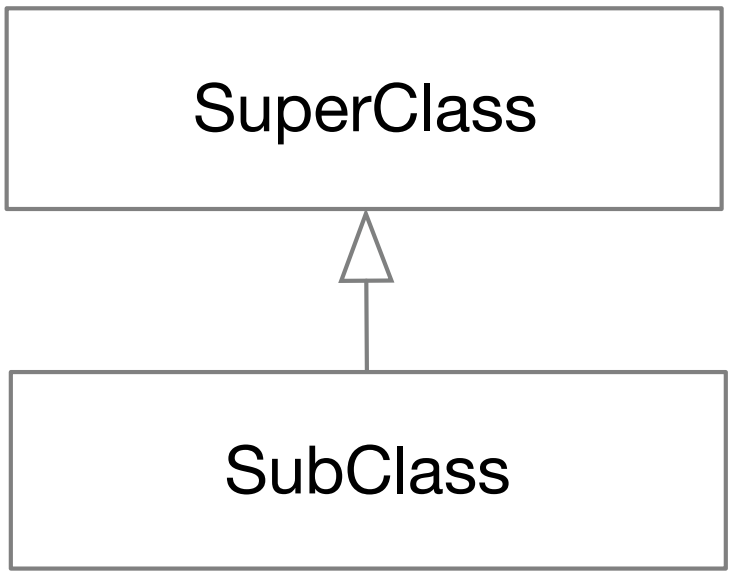
More practice with classes and model-view-controller
(with just a little inheritance)

Logical problem solving: Slide the tiles in the right order

Structure of FiveTwelve



Notation note:



SubClass may inherit or override methods in SuperClass.

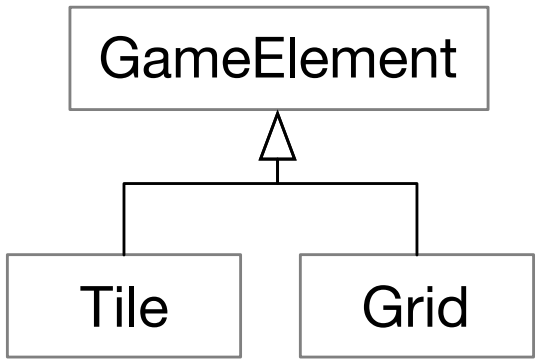
SubClass should be a subtype of SuperClass, meaning it fulfills the contract of SuperClass

Substitution principle: Anywhere SuperClass can be used, SubClass can be used.

NOT guaranteed by Python. The programmer must ensure the substitution principle.

cf: Liskov Substitution Principle

Subclassing in FiveTwelve




```
class Game_Element(object):
    def add_listener(self, listener: GameListener):

        ...
    def notify_all(self, event):
        ...
```

```
class Grid(Game_Element):
  ...
      self.notify_all(
        GameEvent(kind, tile)
```

```
class Tile(Game_Element):
  ...
      self.notify_all(
          GameEvent(kind, self)
```

The ‘add_listener’ and ‘notify_all’ methods are *inherited*.

What is missing

```
# Game moves
def left(self):
    """Slide tiles to the left"""
    movement_vector = ???
    for ???  :
        for ???:
            if tile:
                col.slide(movement_vector)

def right(self):
    """Slide tiles to the right"""
   ...
```

Each move (left, right, up, down) requires checking the tiles in an order consistent with the rules, so they are similar but not identical.

Movement vectors could be [0,1], [1,0], [-1,0], [0,-1] (up, right, left, down)

About those _listeners ...

Model-View-Controller (MVC)
separation of graphics from game logic

"Hooks" for dynamic binding

```
class Listener(object):
    """Abstract base class --- defines required behavior"""
    def notify(self, msg: str):
        """All subclasses must implement 'notify' """
        raise NotImplementedError(
                "Class {} must implement 'notify' method".format(type(self)))


class Dog(object):
    def __init__(self, name: str):
        self.name = name
        self._listeners = [ ]

    def add_listener(self, listener: Listener):
        self._listeners.append(listener)

    def notify_all(self, speech: str):
        for listener in self._listeners:
                listener.notify("{} says {}".format(self.name, speech))
```

A method in 'Dog' invokes the notification

```
class Dog(object):
    """Might as well"""

    def __init__(self, name: str):
        self.name = name
        self._listeners = [ ]

    def add_listener(self, listener: Listener):
        self._listeners.append(listener)

    def notify_all(self, speech: str):
        for listener in self._listeners:
            listener.notify("{} says {}".format(self.name, self.speech))
```

```
def speak(self, msg: str):
    self.notify_all(msg)
```

Define a concrete listener (subclassing the abstract listener)

```
class SpeaksInStars(Listener):

    def notify(self, msg: str):
        star_bar = (8 + len(msg)) * '*'
        print(star_bar)
        print("*** {} ***".format(msg))
        print(star_bar)
```

Attach a listener to a Dog

```
class Dog(object):
    ...
    def notify_all(self, speech: str):
        for listener in self._listeners:
            listener.notify("{} says {}".format(self.name, self.speech))

    def speak(self, msg: str):
        self.notify_all(msg)


class SpeaksInStars(Listener):
    def notify(self, msg: str):
            ...
```

```
fido = Dog("Fido")
star_speaker = SpeaksInStars()
fido.add_listener(star_speaker)
fido.speak(”dogs can't really talk”)
```

What is the output?

Why all this trouble to call a method?

It's all about dependence!

I do want Dog to notify (call) a listener

But not a particular listener ...

Dog should not depend on SpeaksInStars.

I can attach any subclass of Listener to Dog

Why all this trouble to call a method?

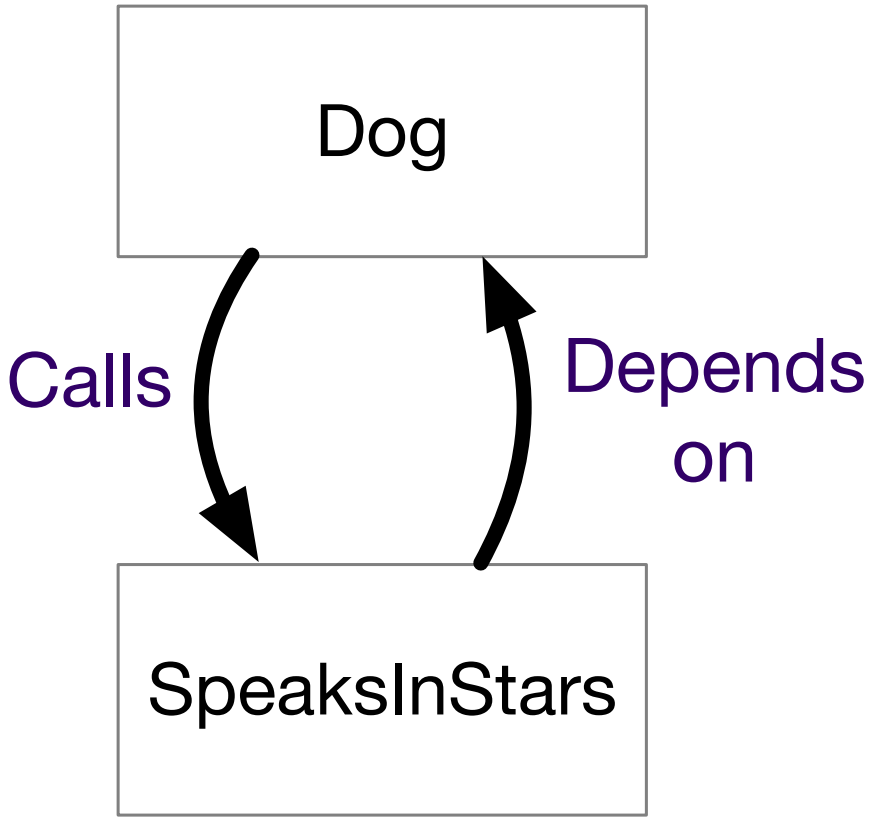
It's all about dependence!

I do want Dog to notify (call) a listener

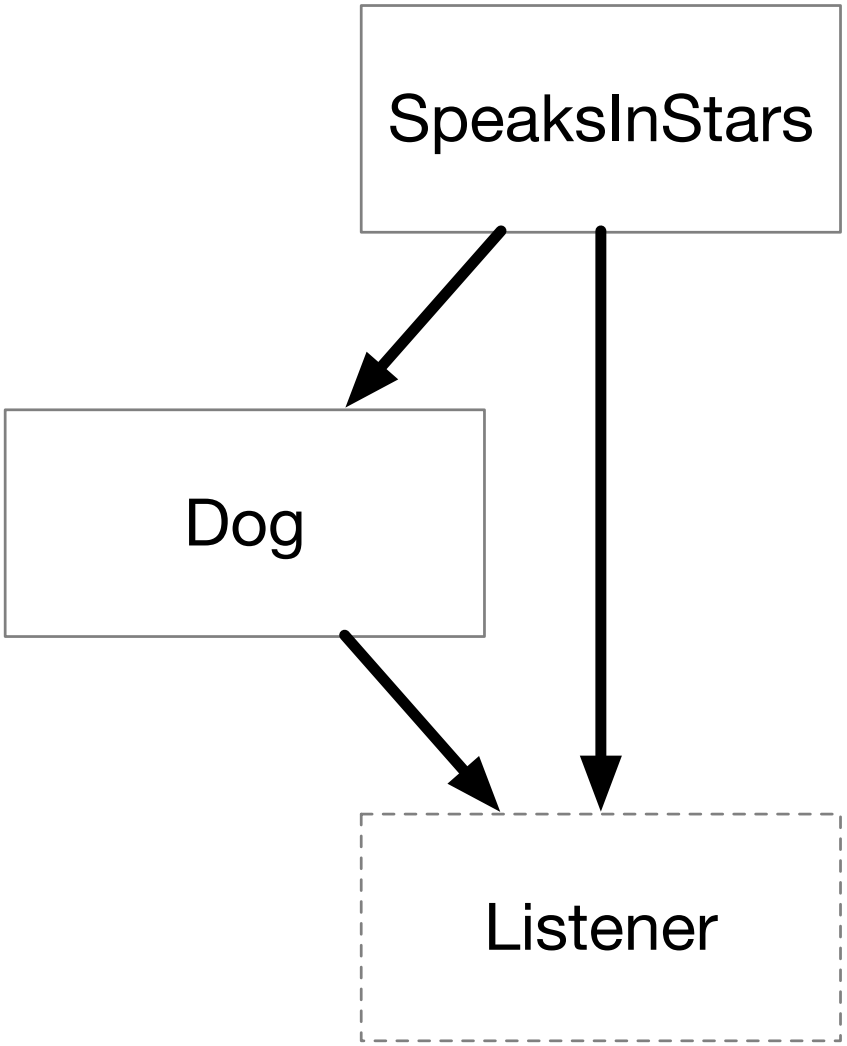
But not a particular listener ...

Dog should not depend on SpeaksInStars.

I can attach any subclass of Listener to Dog



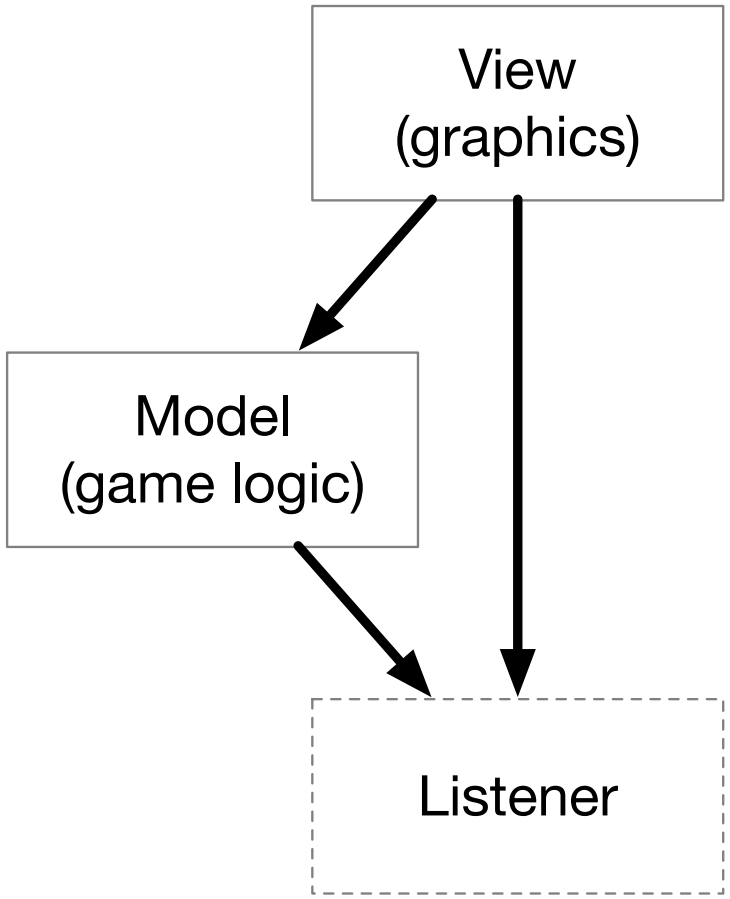
Dependence



Concrete listener class:
what this listener does

Abstract base class:
what every listener must do

Model-View-Controller



I don't want the game logic to depend on the graphics ...

I should be able to change the graphics package without changing the game logic (model component) at all.

So the FiveTwelve "model" just notifies listeners of events that might trigger changes in the graphics.