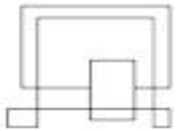
VIA University College



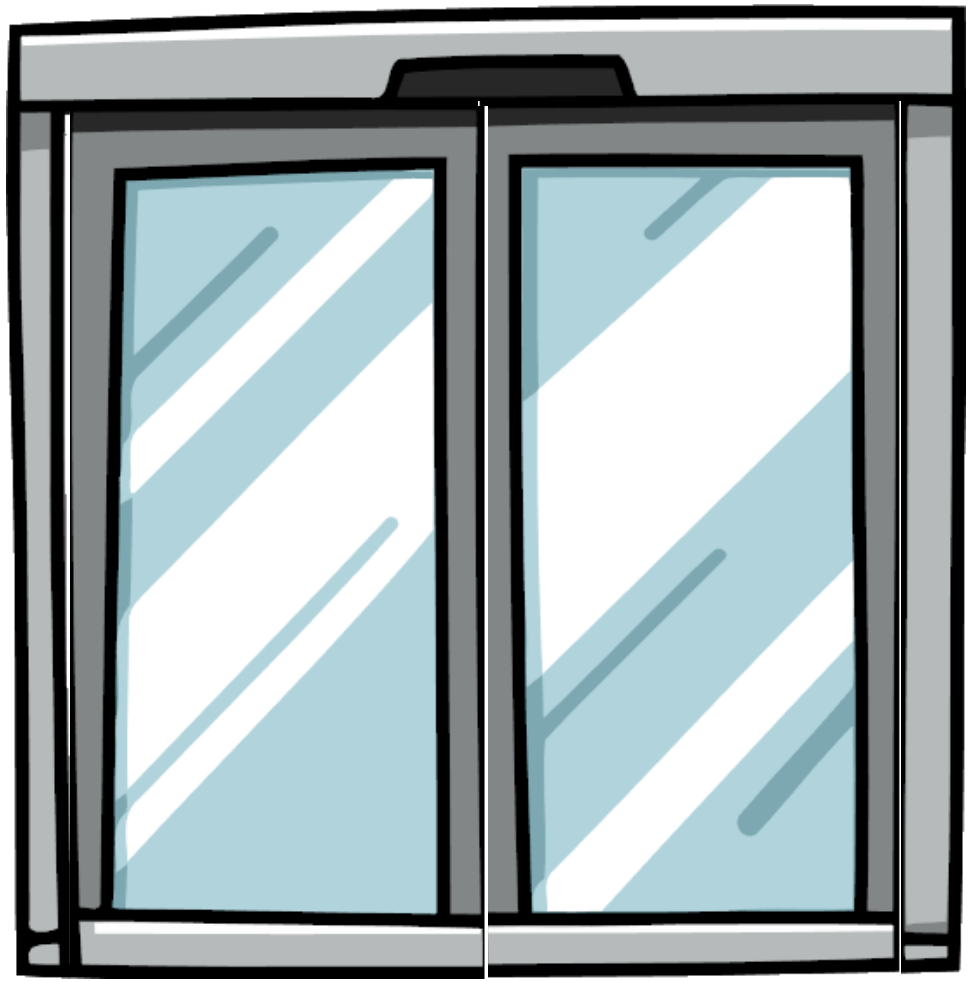
Software Development with UML and Java 2

State pattern

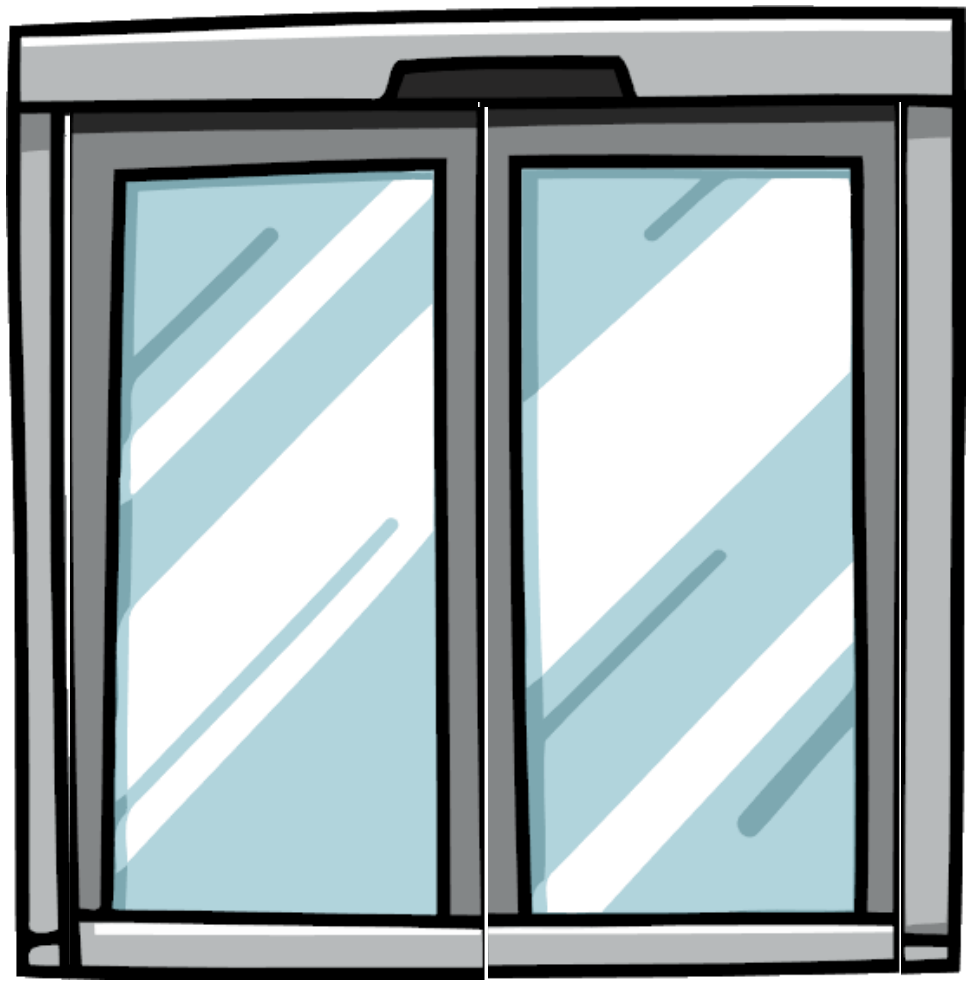
Agenda

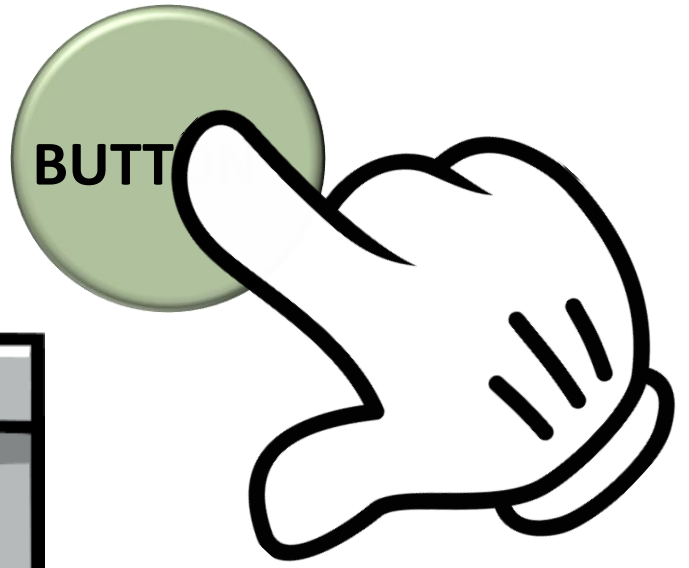
- Concept by example
  - Automatic doors
  - Radiator
  - Mobile phone
  - Animation/controls in games
  - Threads
- Naïve approach
- State pattern
  - What is the purpose?
  - UML structure
- How to apply the state pattern?

Automatic door example

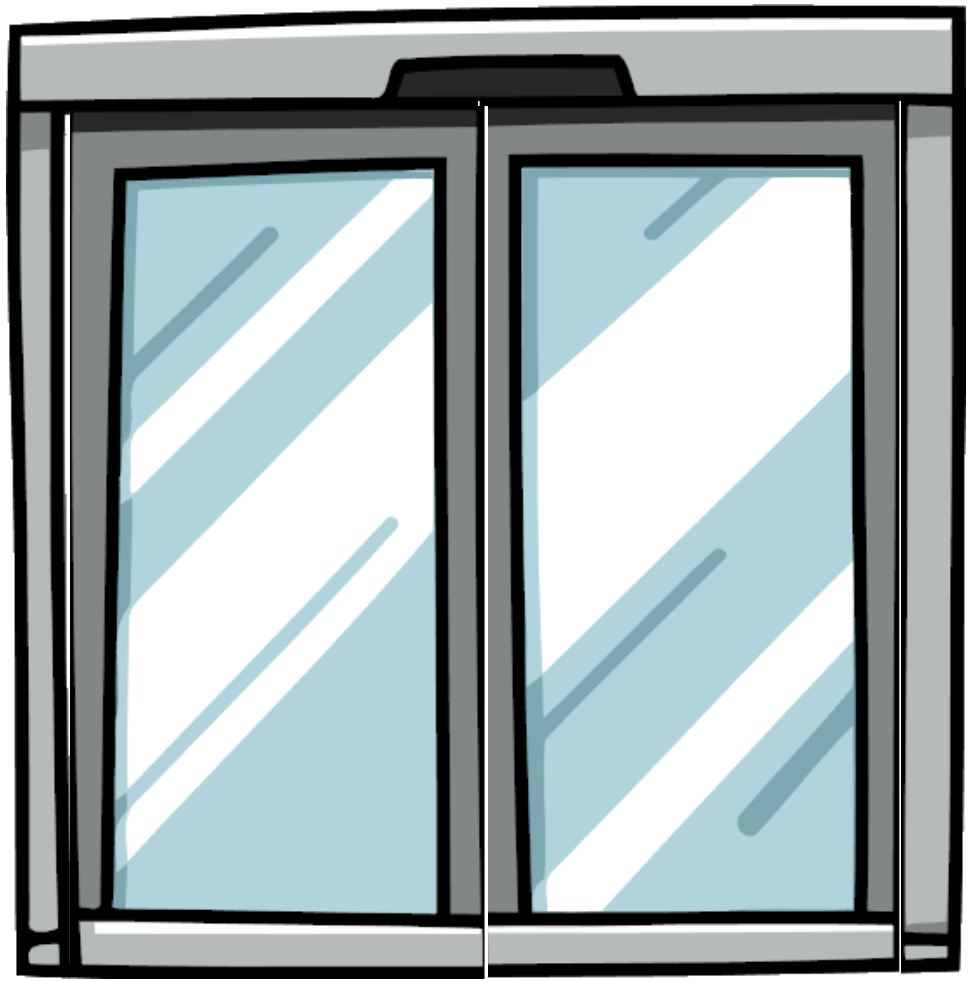


Automatic door example

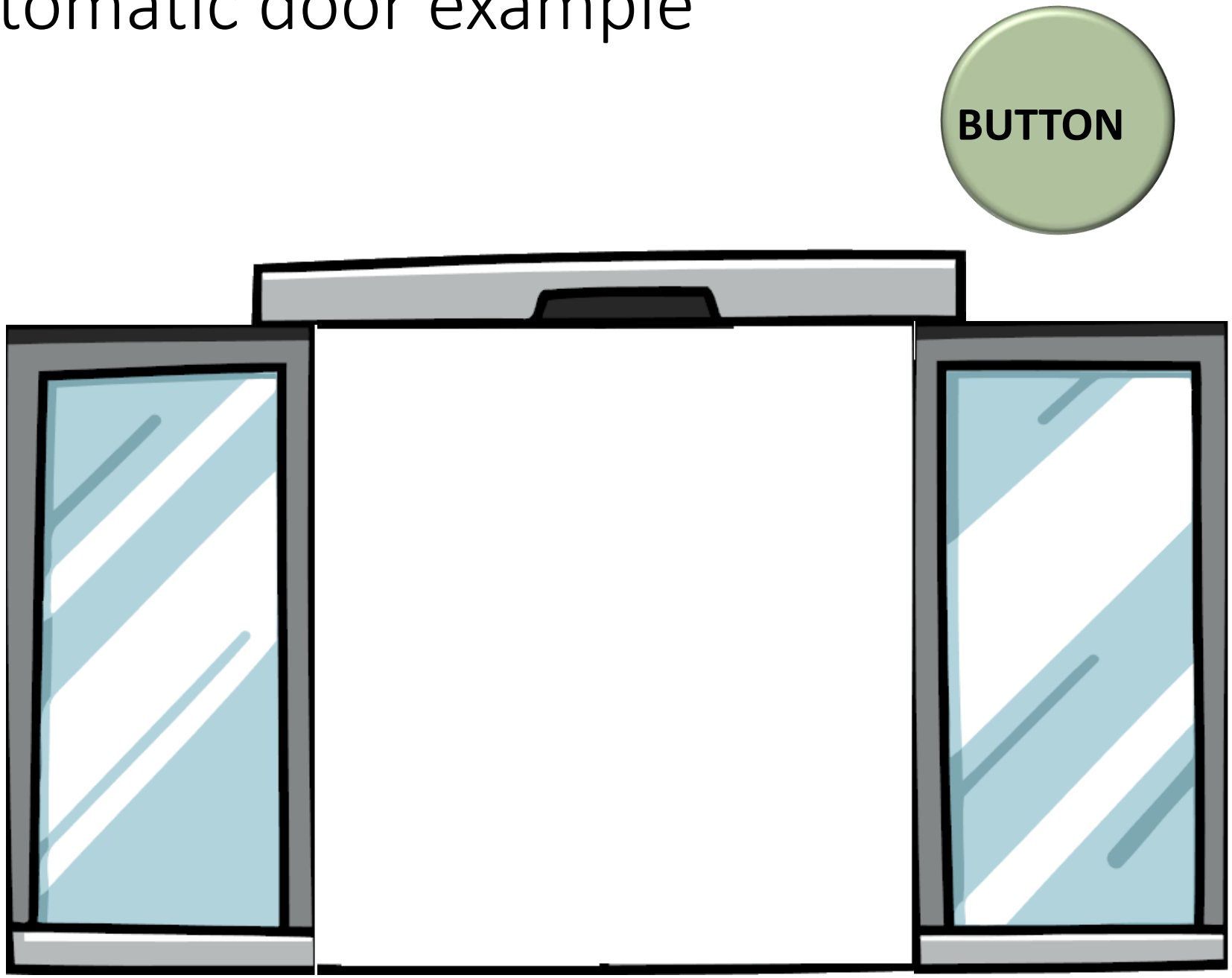




Automatic door example



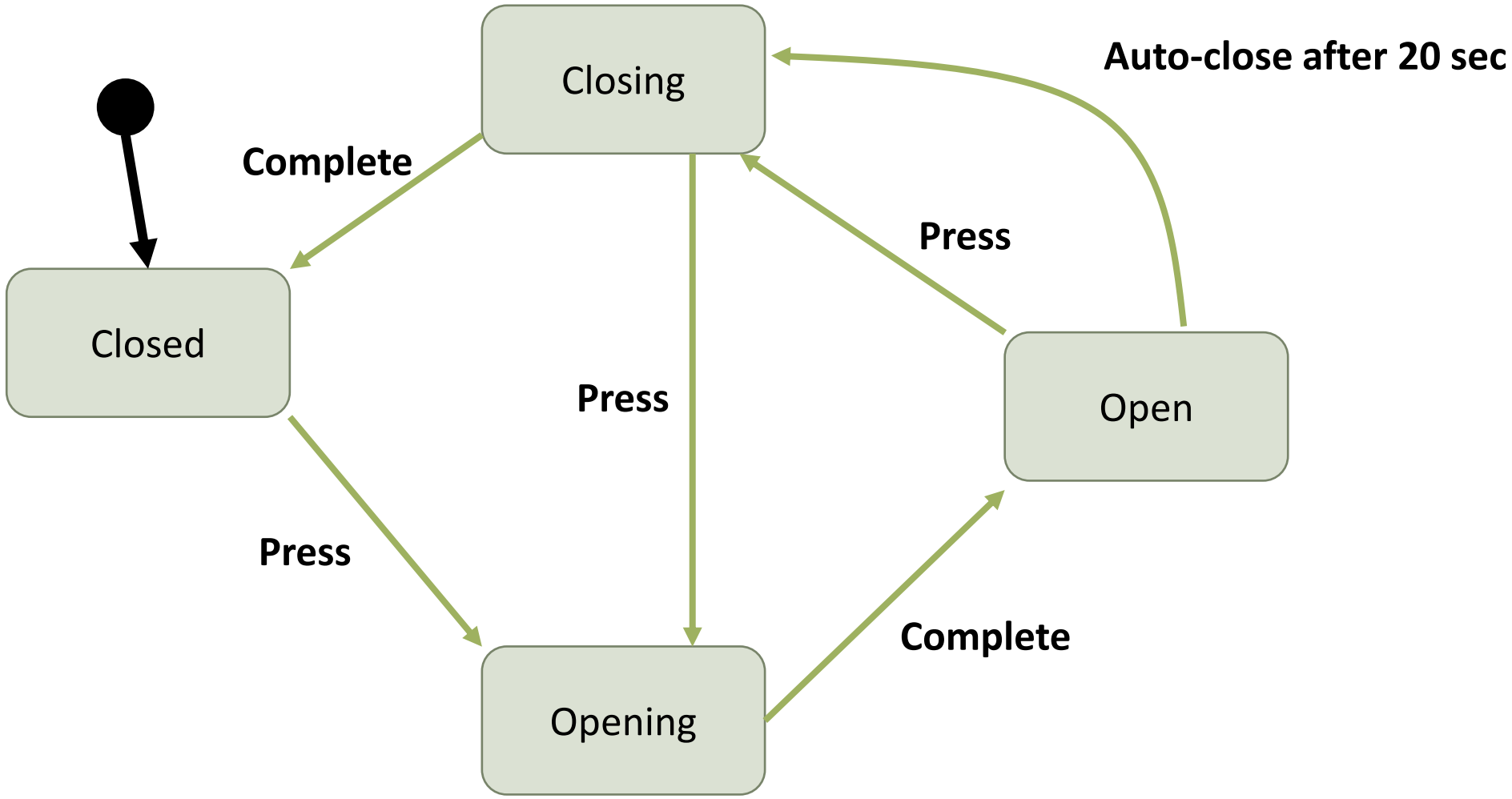
Automatic door example



Door states

- When closed, Click to open
- When open, click to close
- While closing, click to open
- If open, after 20 seconds, close

Door state machine diagram



- My door object can be in different states
- Based on the state, clicking the button results in different behavior.

- My door changes behavior based on its state

Agenda

- Concept by example
  - Automatic doors
  - Radiator
  - Mobile phone
  - Animation/controls in games
  - Threads
- Naïve approach
- State pattern
  - What is the purpose?
  - UML structure
- How to apply the state pattern?

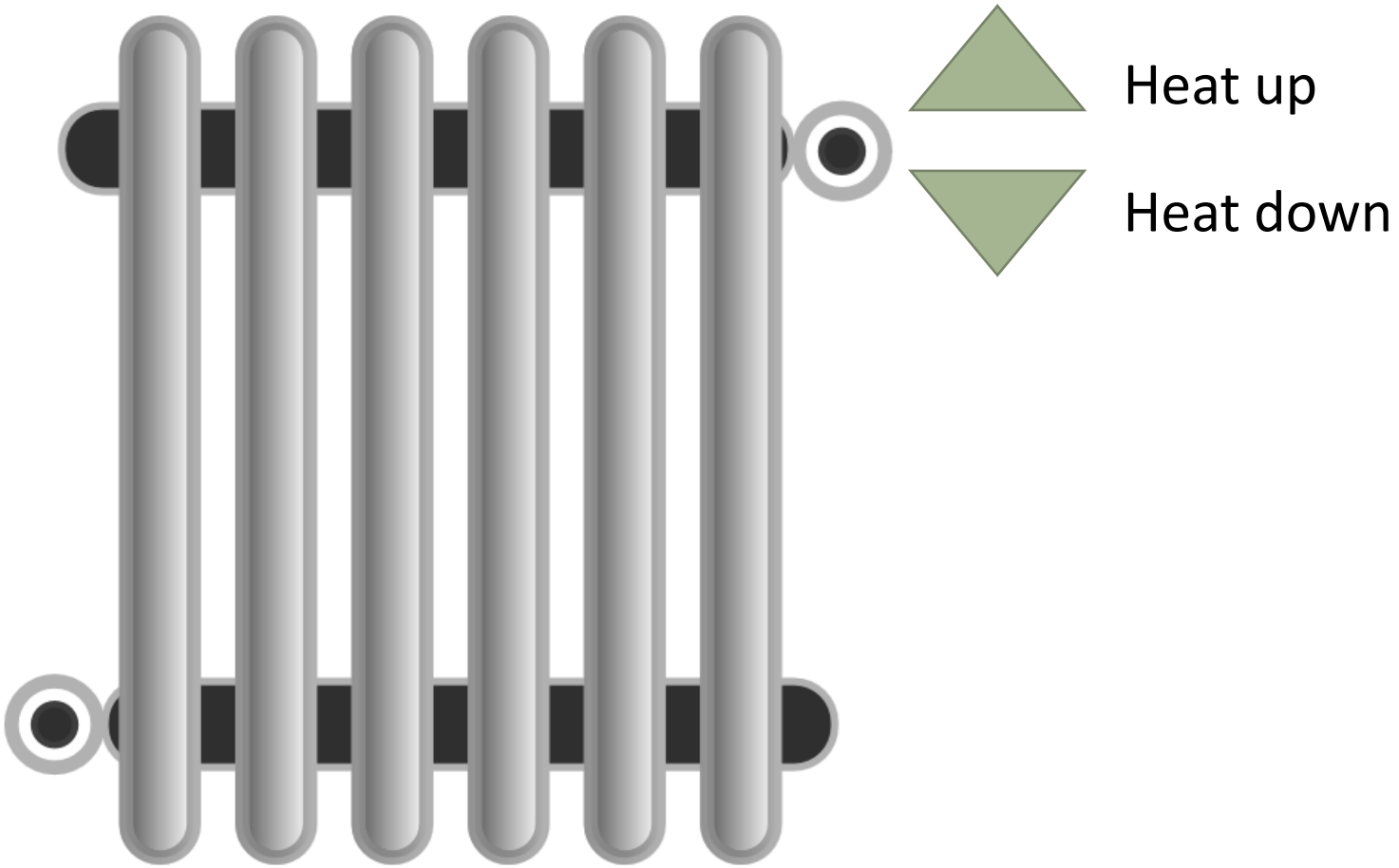
Radiator

3

2

1

Off



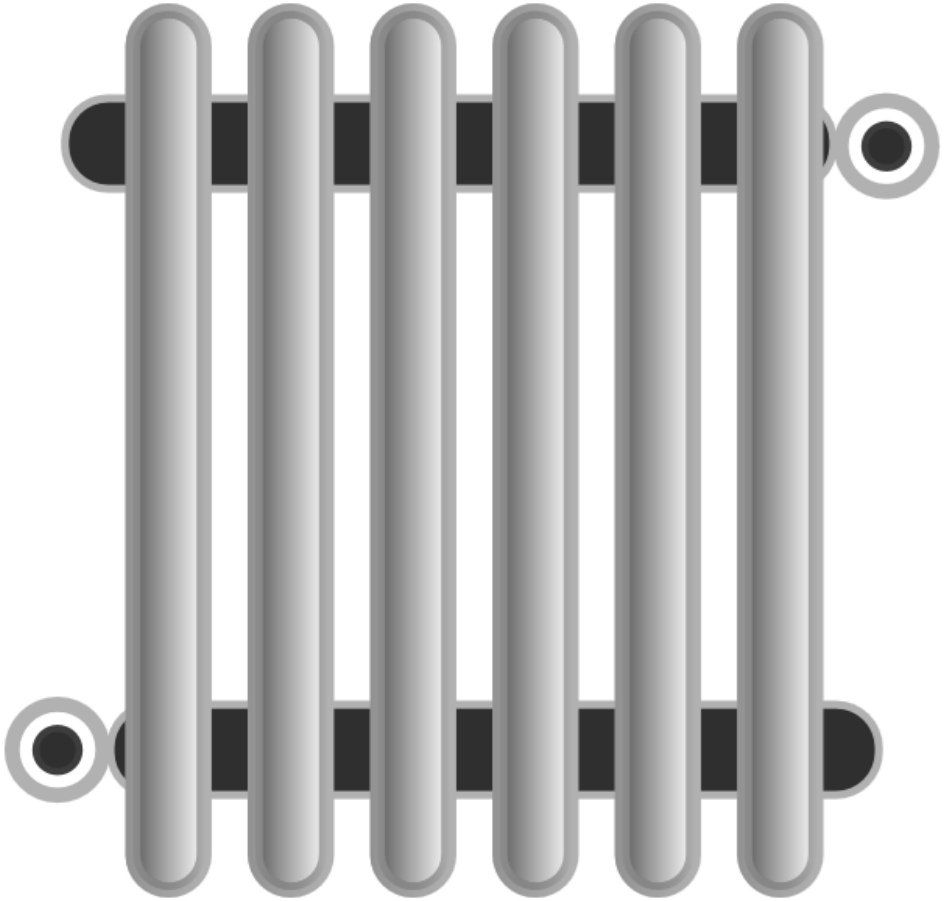
Radiator

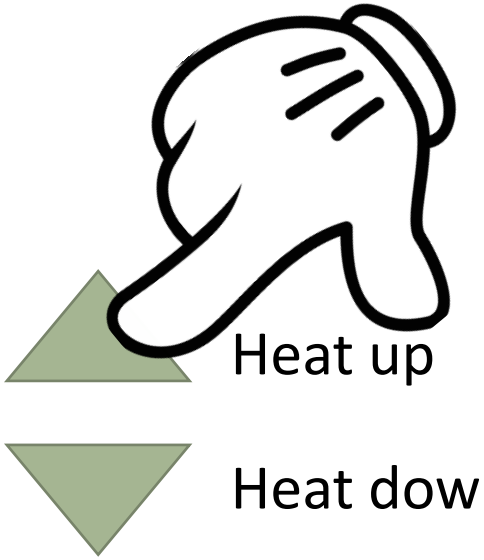
3

2

1

Off





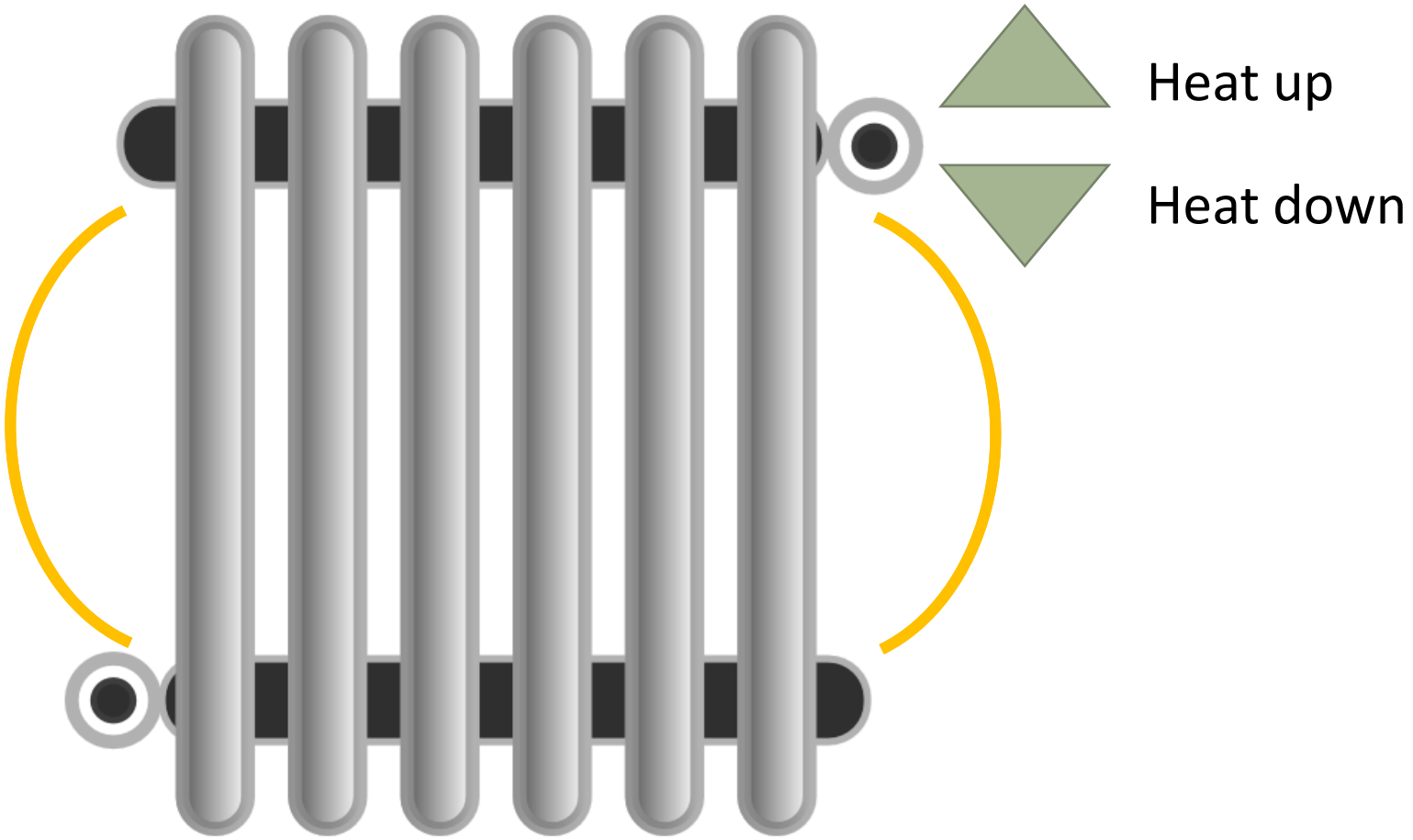
Radiator

3

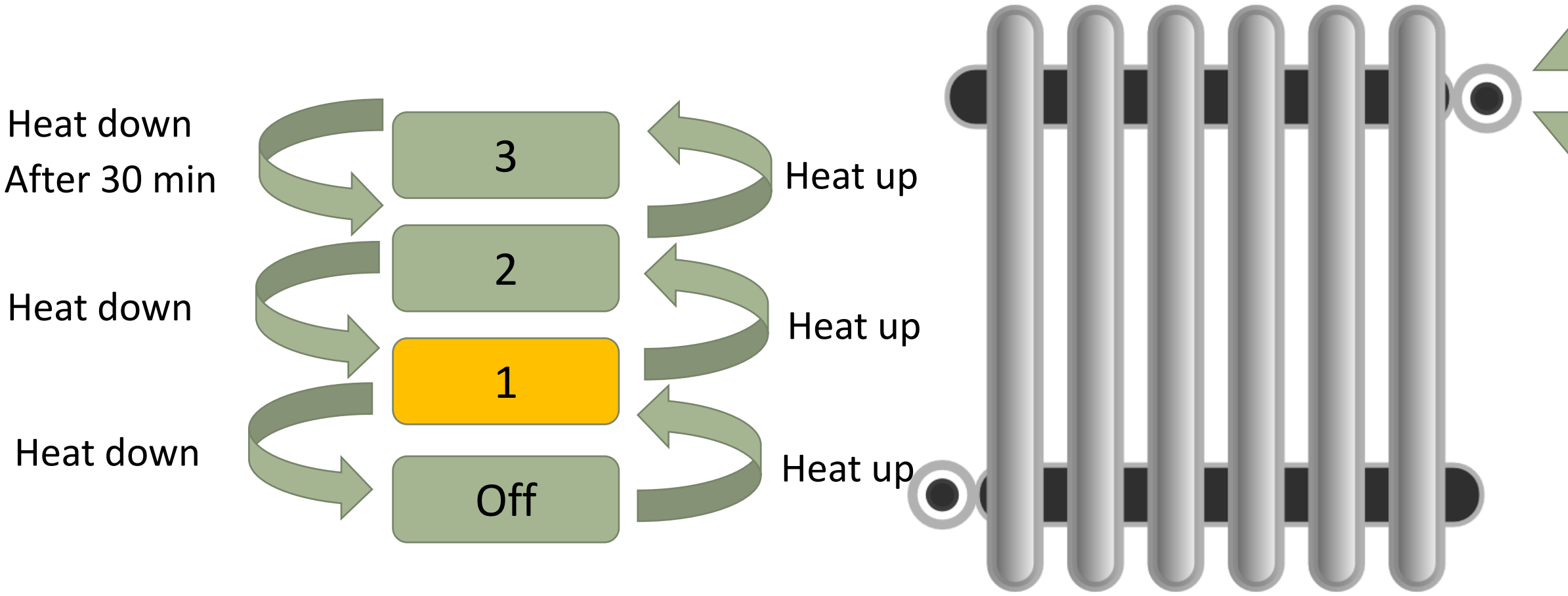
2

1

Off



Radiator

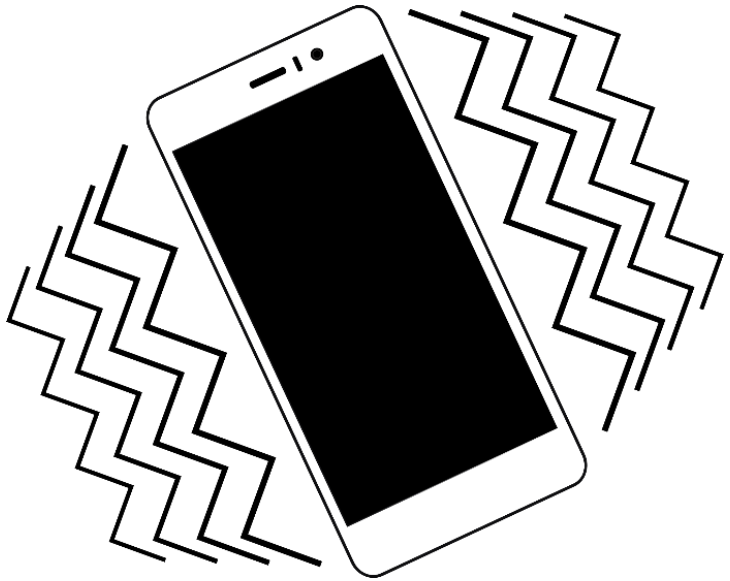

Heat down
After 30 min
Heat down
Heat down
3
2
1
Off
Heat up
Heat up
Heat up

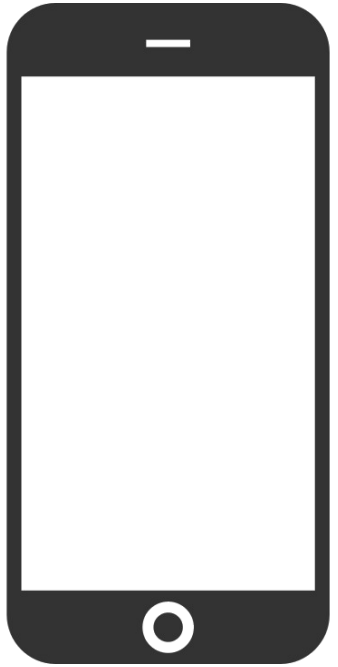
Agenda

- Concept by example
  - Automatic doors
  - Radiator
  - Mobile phone
  - Animation/controls in games
  - Threads
- Naïve approach
- State pattern
  - What is the purpose?
  - UML structure
- How to apply the state pattern?

A mobile phone

- Events (methods called on my phone):
  - receive a message/call
  - Press volume up/down
- States:
  - Ring, Vibrate, silent



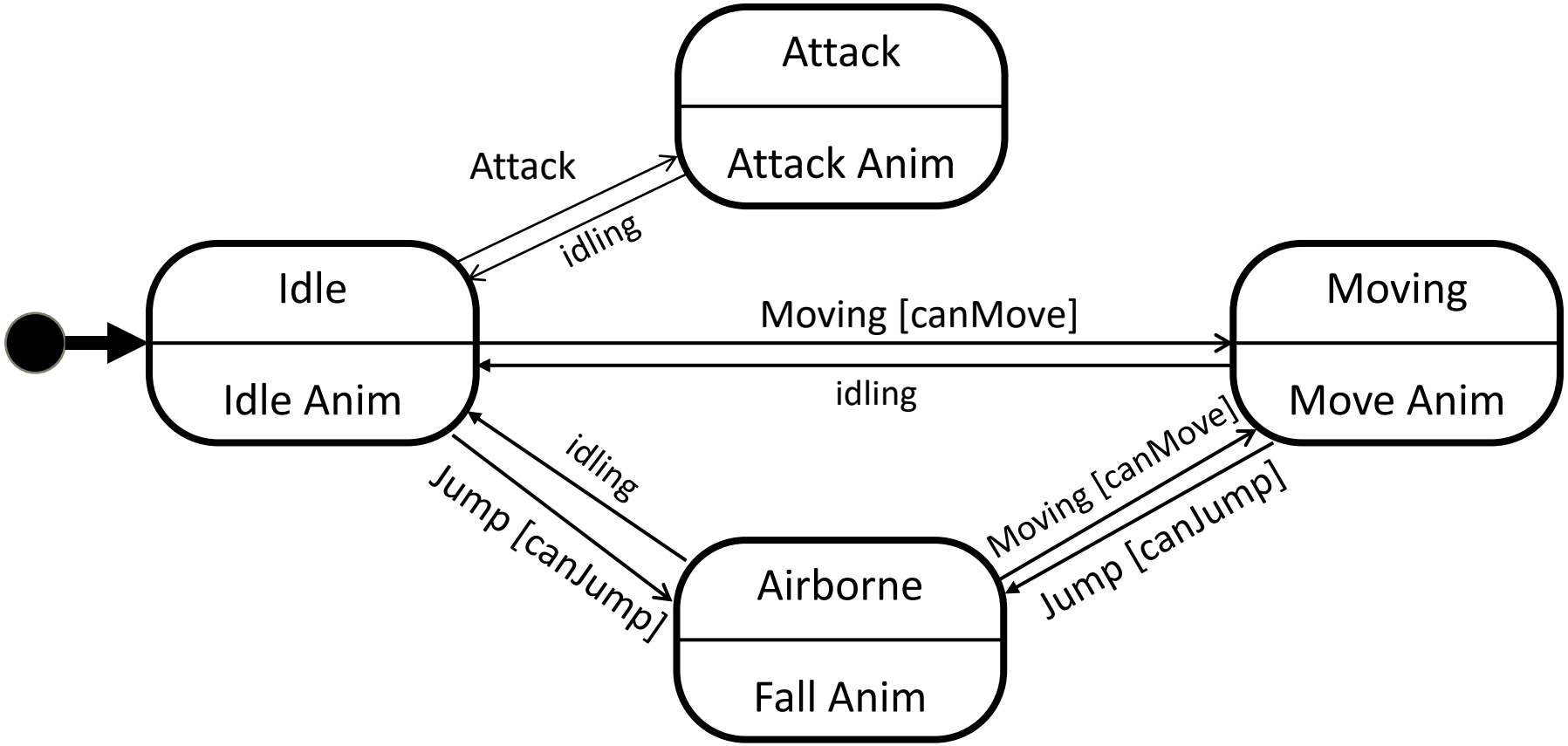


Agenda

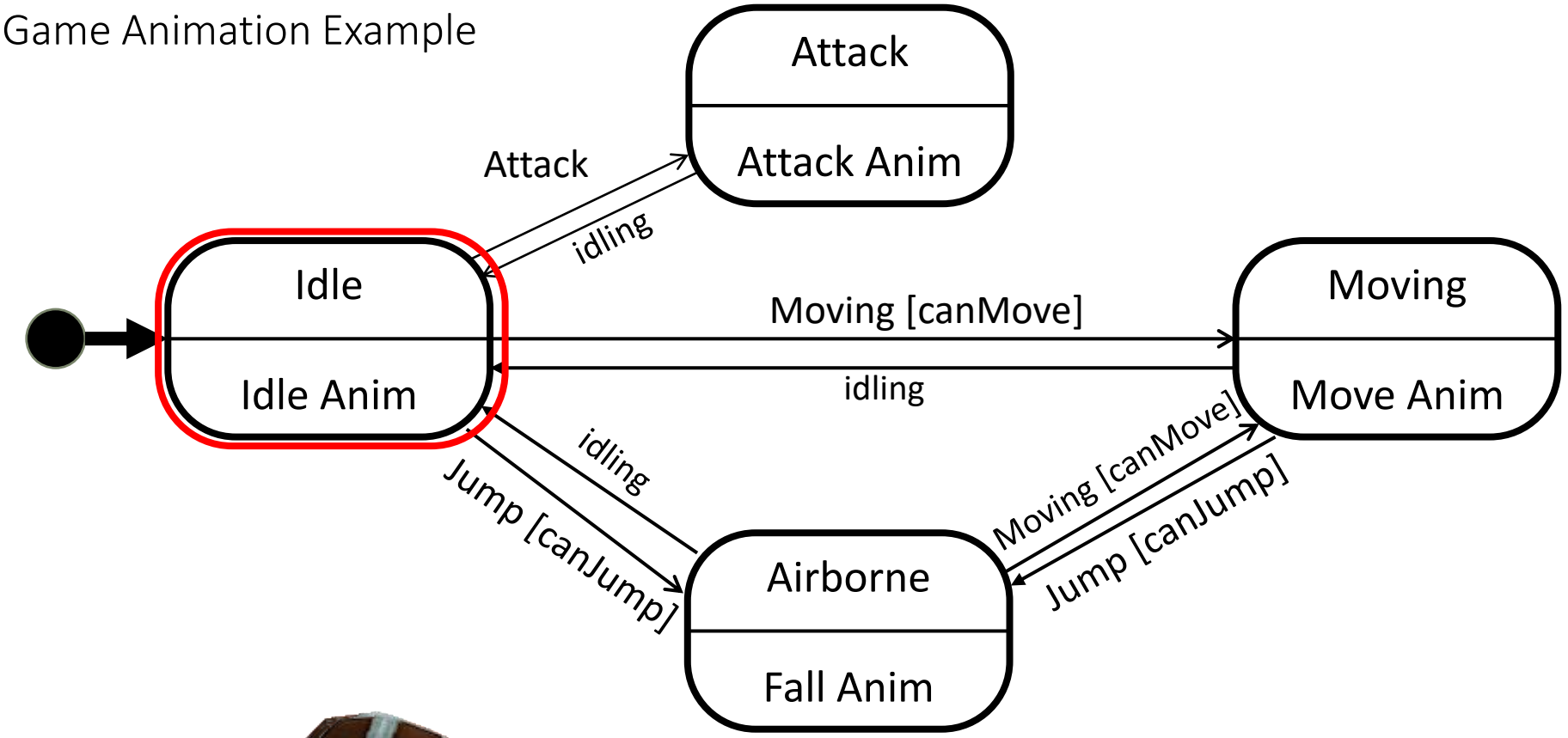
- Concept by example
  - Automatic doors
  - Radiator
  - Mobile phone
  - Animation/controls in games
  - Threads
- Naïve approach
- State pattern
  - What is the purpose?
  - UML structure
- How to apply the state pattern?

Animations in games

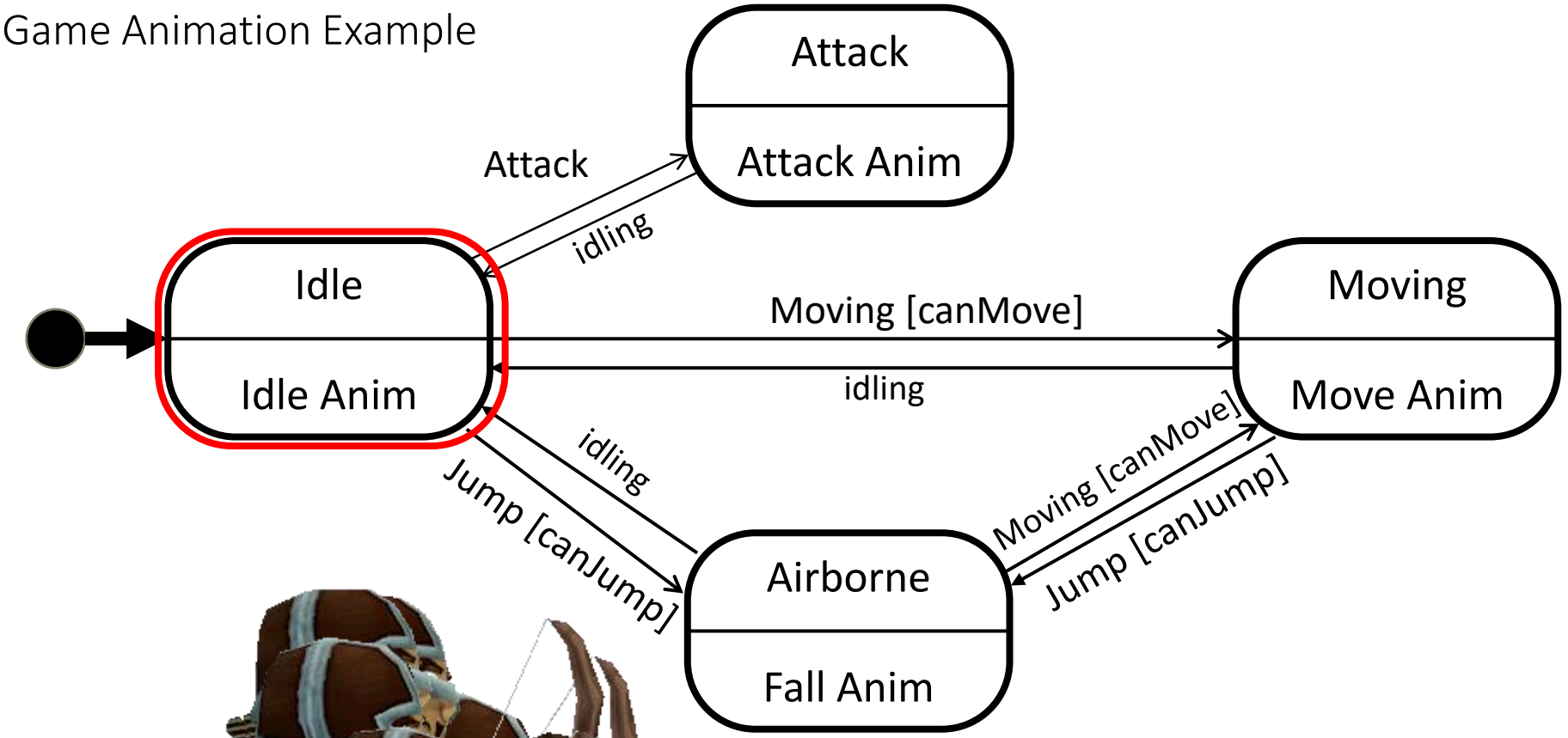
Game Animation Example



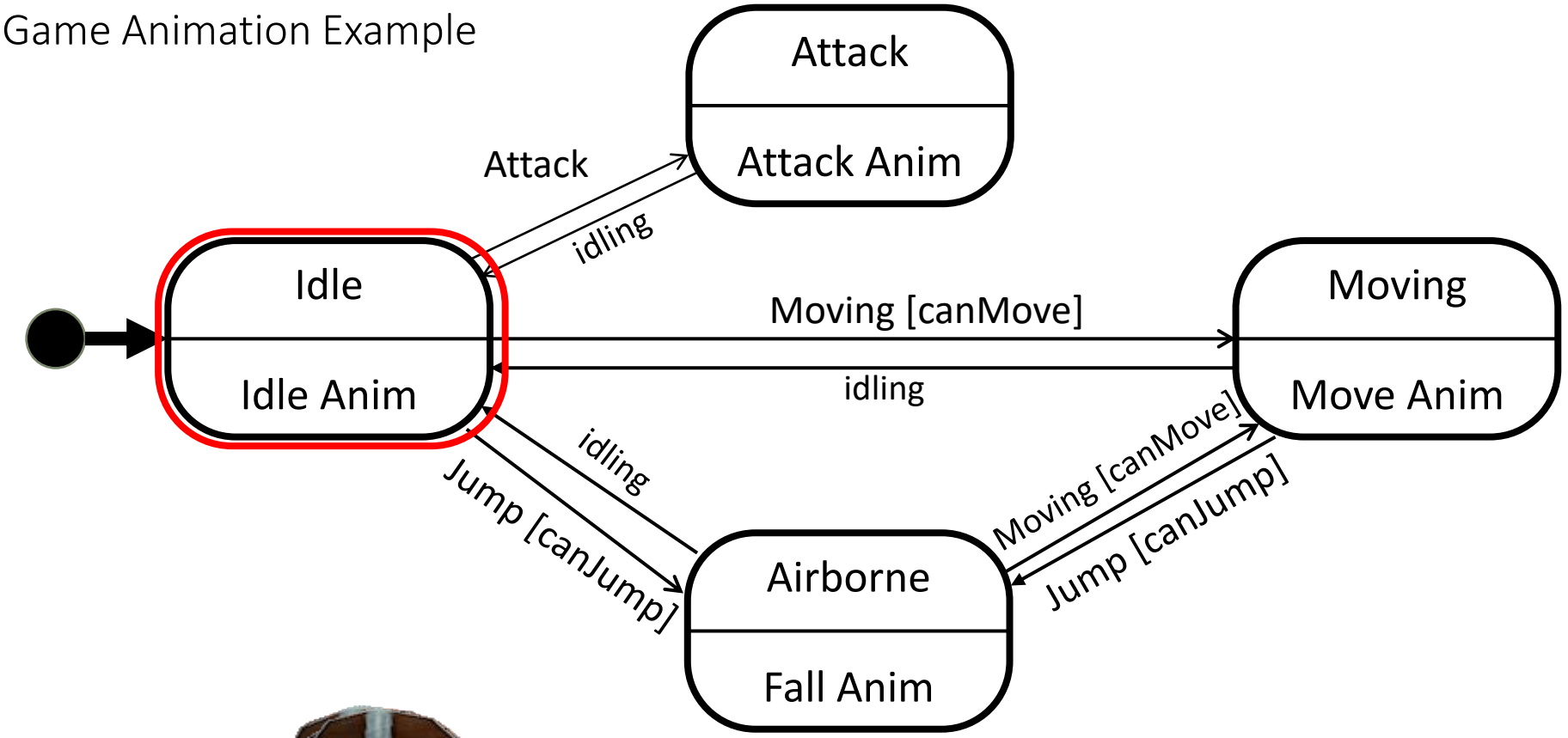
Game Animation Example



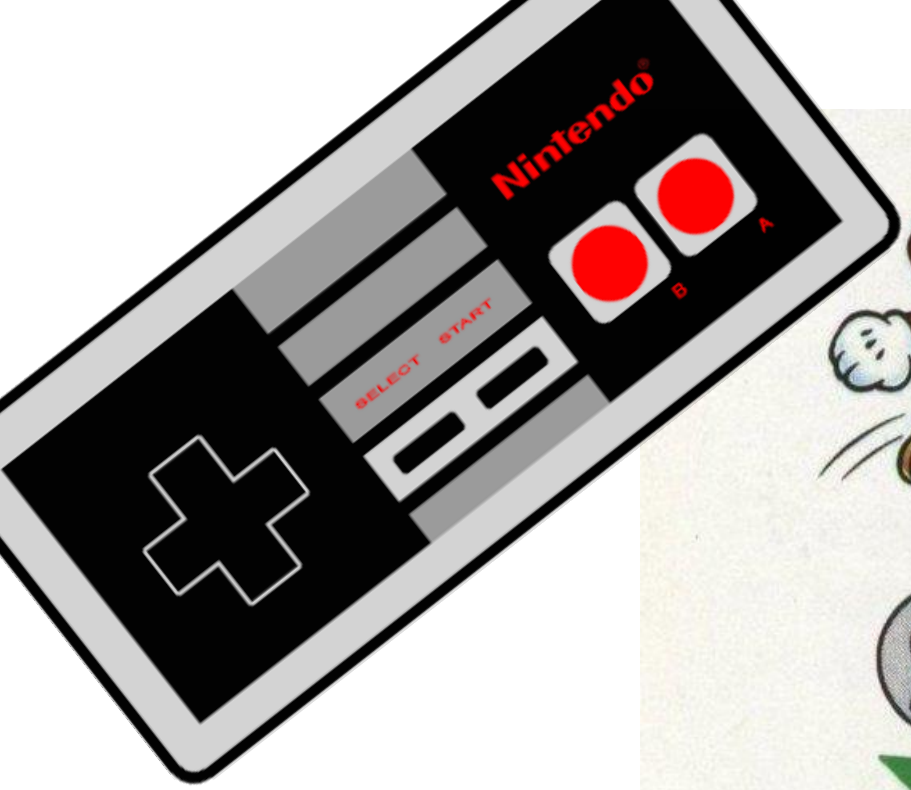
Game Animation Example

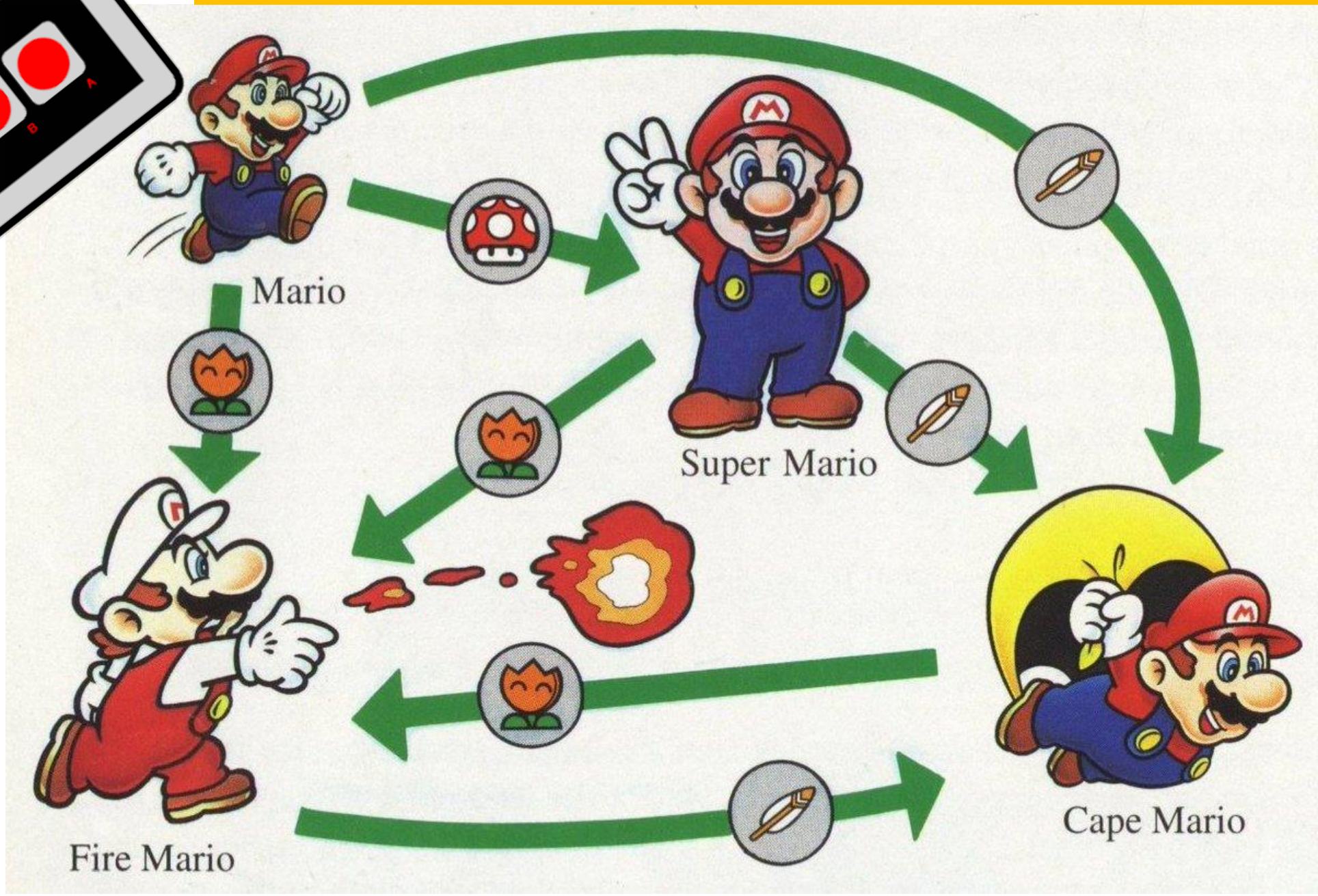


Game Animation Example



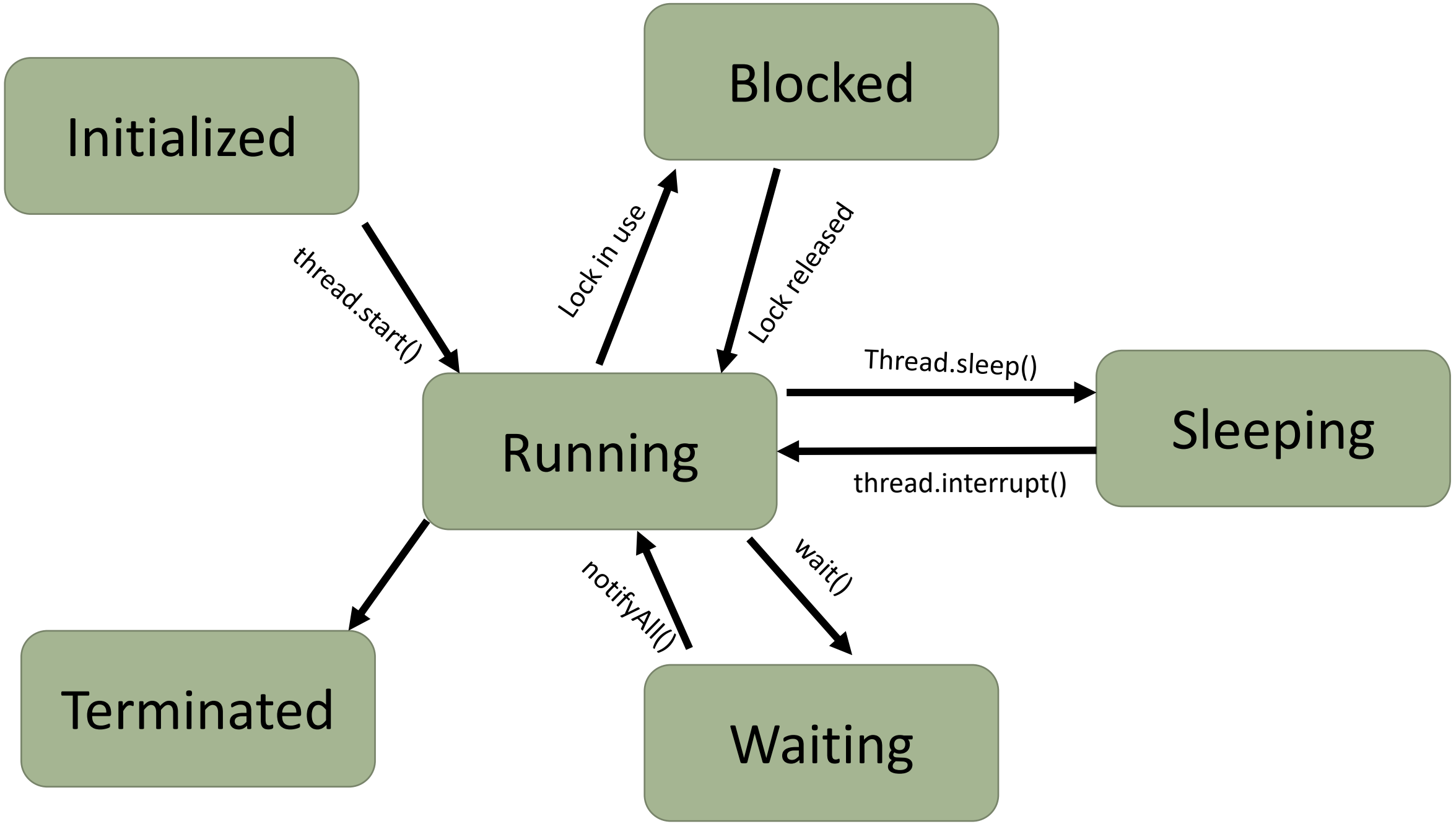
Controlling your characters behavior based on states





Agenda

- Concept by example
  - Automatic doors
  - Radiator
  - Mobile phone
  - Animation/controls in games
  - Threads
- Naïve approach
- State pattern
  - What is the purpose?
  - UML structure
- How to apply the state pattern?

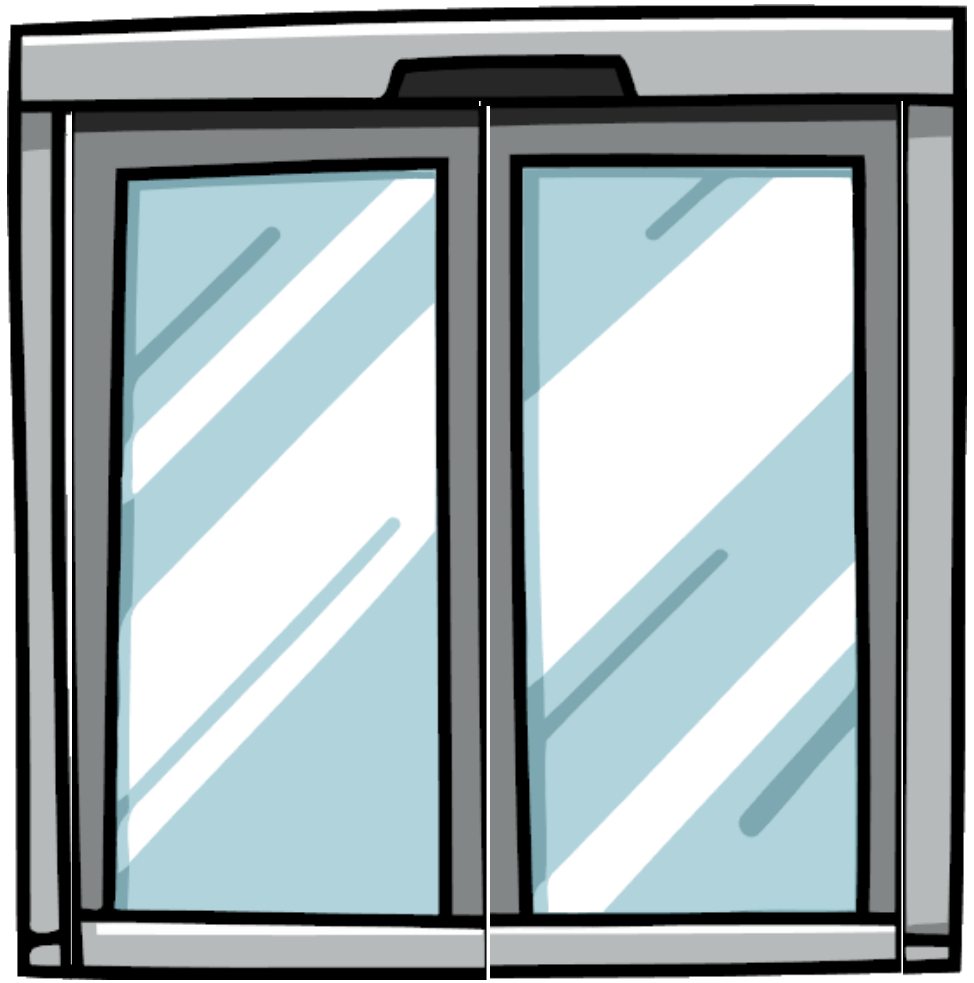

Initialized
Blocked
thread.start()
Lock in use
Lock released
Thread.sleep()
Running
Sleeping
thread.interrupt()
Terminated
notifyAll()
wait()
Waiting

Agenda

- Concept by example
  - Automatic doors
  - Radiator
  - Mobile phone
  - Animation/controls in games
  - Threads
- Naïve approach
- State pattern
  - What is the purpose?
  - UML structure
- How to apply the state pattern?

Automatic door example

When closed, Click to open
When open, click to close
While closing, click to open
If open, after 20 seconds -> close



How do we implement this?

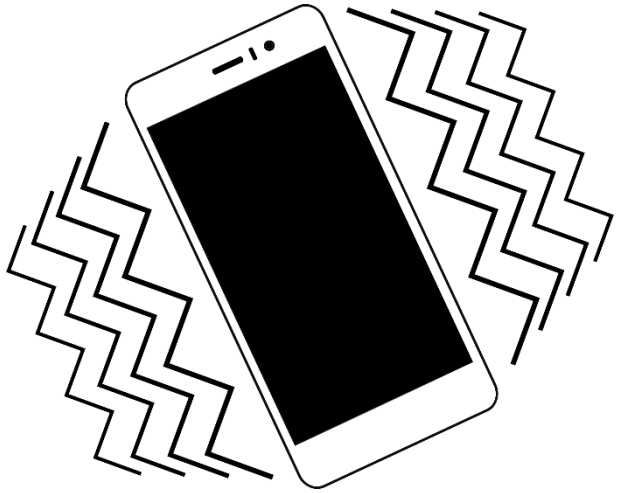
- A bunch of if statements?

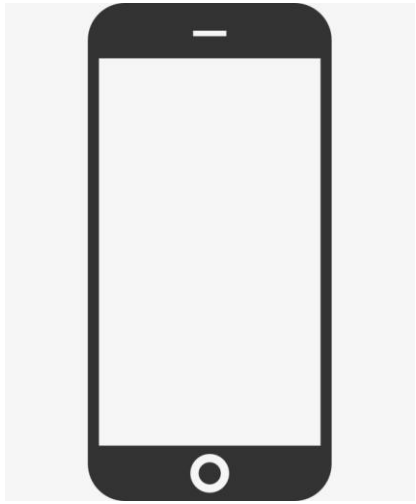
```
Public void clickButton() {
    if(isClosed())
        if(buttonPushed())
            open()
    else if(isOpening())
        If(buttonPushed())
            // do nothing
        else if (complete())
            // do nothing
    else if(isOpen())
        If(buttonPushed())
            stayOpen()
    ….
```

Too many cases quickly become confusing

A mobile phone

- Events:
  - receive a message/call
  - Press volume up/down
  - Change state (alert slider?)
- States:
  - Ring, Vibrate, silent





```
public class Phone {

    private boolean isOnSilent, isOnVibrate, isOnSound = true;

    public void receiveMessage(String txt) {
        if(isOnSilent) {
            // nothing
        } else if(isOnSound) {
            beepBeep();
        } else if(isOnVibrate) {
            vibrate();
        }
        System.out.println("Message received:");
        System.out.println(txt);
    }

    public void receiveCall() {
        if(isOnSilent) {
```

```
public void receiveCall() {
    if(isOnSilent) {
        lightUpScreen();
    } else if(isOnSound) {
        playRingTone();
    } else if(isOnVibrate) {
        vibrate();
    }
}

public void volumeUpButton() {
    if(isOnSound) {
        turnVolumeUp();
    } else if(isOnVibrate) {
        // change to sound
        isOnSound = true;
        isOnVibrate = false;
    } else if(isOnSilent) {
```

```
}

public void volumeUpButton() {
    if(isOnSound) {
        turnVolumeUp();
    } else if(isOnVibrate) {
        // change to sound
        isOnSound = true;
        isOnVibrate = false;
    } else if(isOnSilent) {
        isOnSilent = false;
        isOnSound = true;
    }
}
```

Again, can become complicate with the number of conditions and variables to consider

- I can have a (potentially) large number of if-statements
- Expanding the system, requires expanding the sequence of if-statements, perhaps multiple places.
- Can quickly become chaotic with too many if-statements, and variables to keep track of.

Agenda

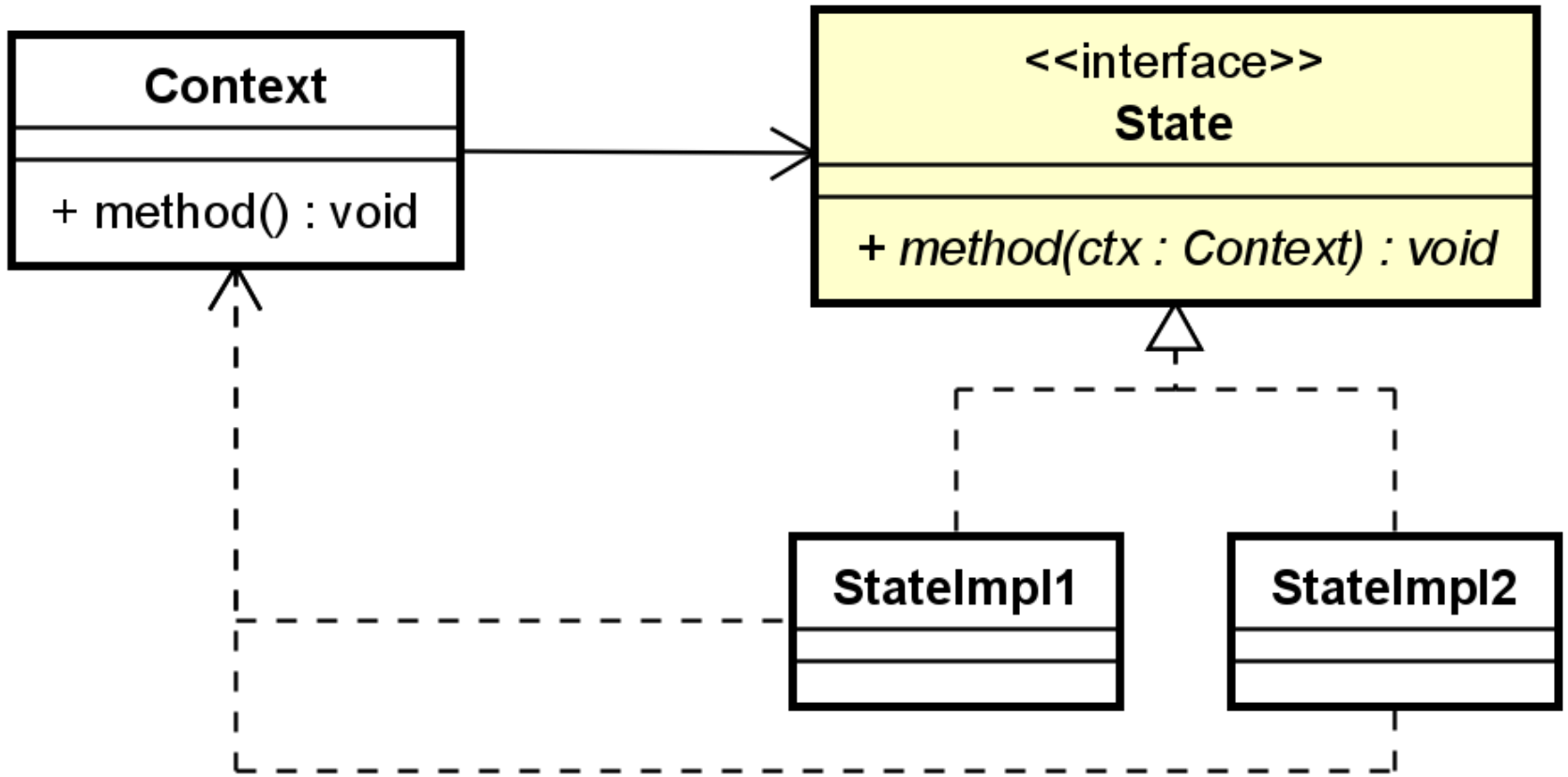
- Concept by example
  - Automatic doors
  - Radiator
  - Mobile phone
  - Animation/controls in games
  - Threads
- Naïve approach
- Study group
- State pattern
  - What is the purpose?
  - UML structure
- How to apply the state pattern?

State pattern purpose

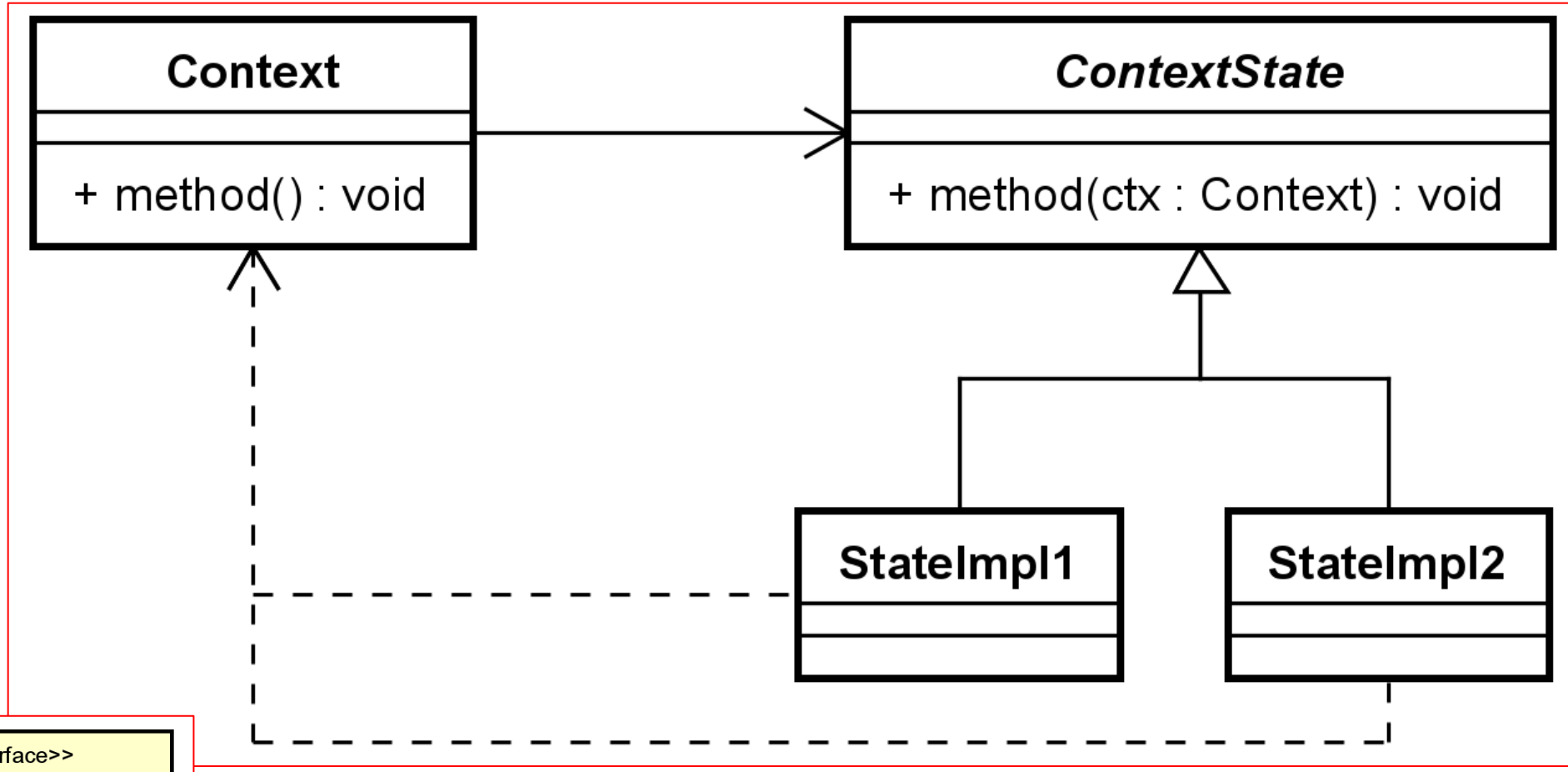
- Objects changes internal behavior, based on internal state.
- We want to encapsulate these state-specific behaviors in separate classes
- A class per possible state

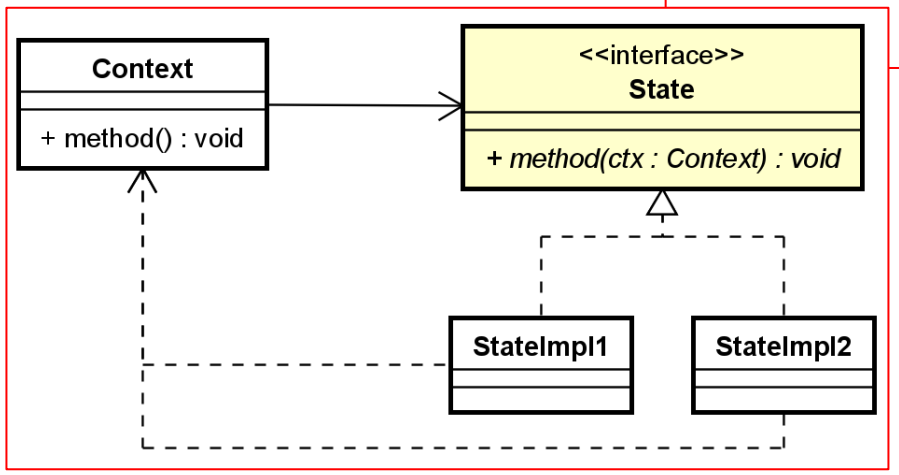
- This removes all the if-statements, and conditional checking.
- Input to the class (method calls) are handled by the state classes.

General UML


Context
+ method() : void
<<interface>>
State
+ method(ctx : Context) : void
StateImpl1
StateImpl2

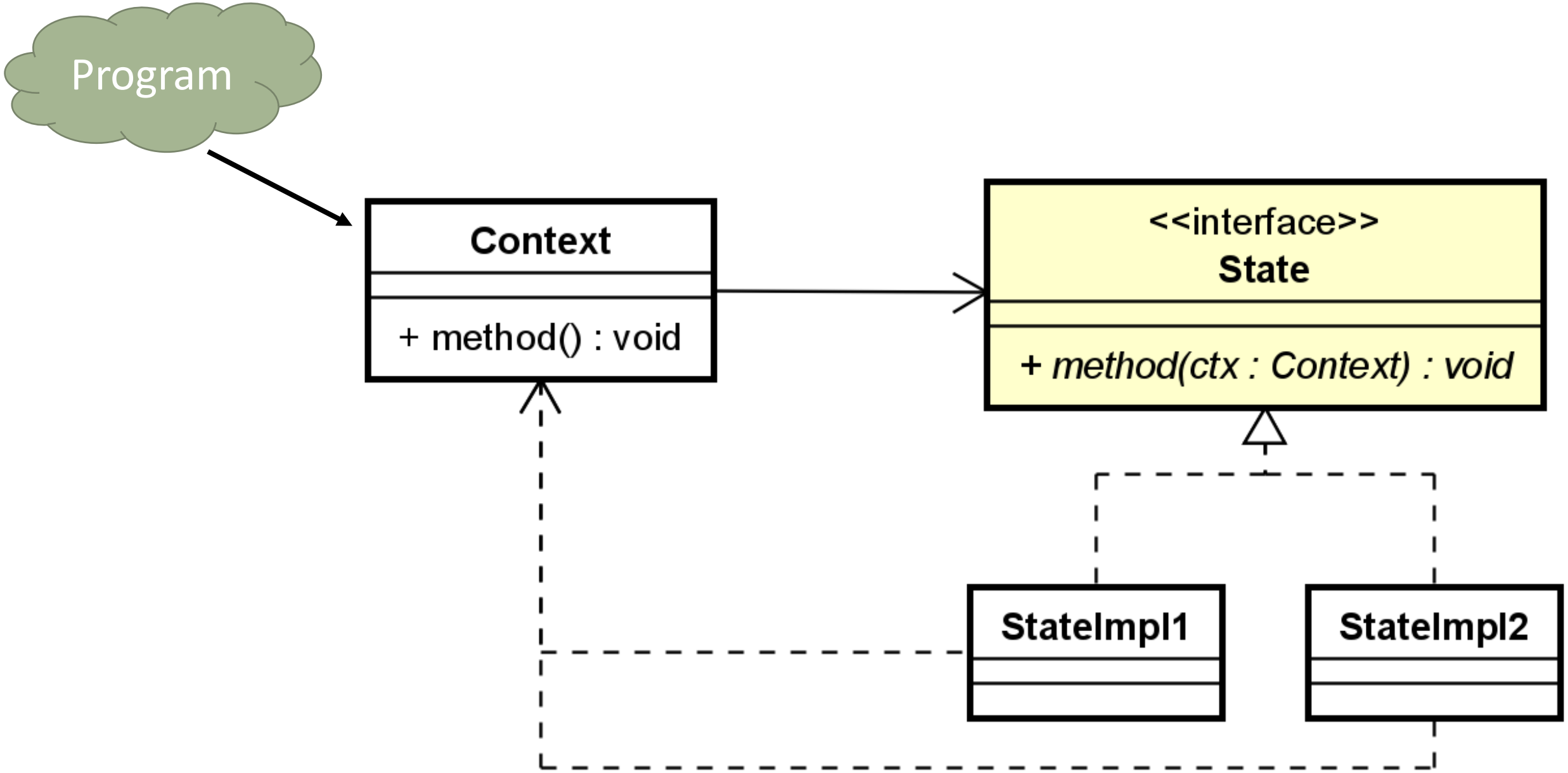
General UML

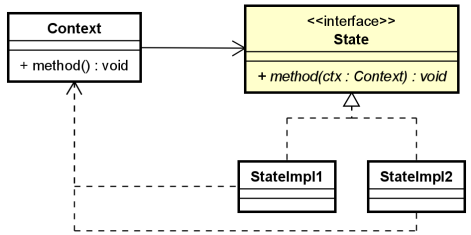

Context
+ method() : void
ContextState
+ method(ctx : Context) : void
StateImpl1
StateImpl2

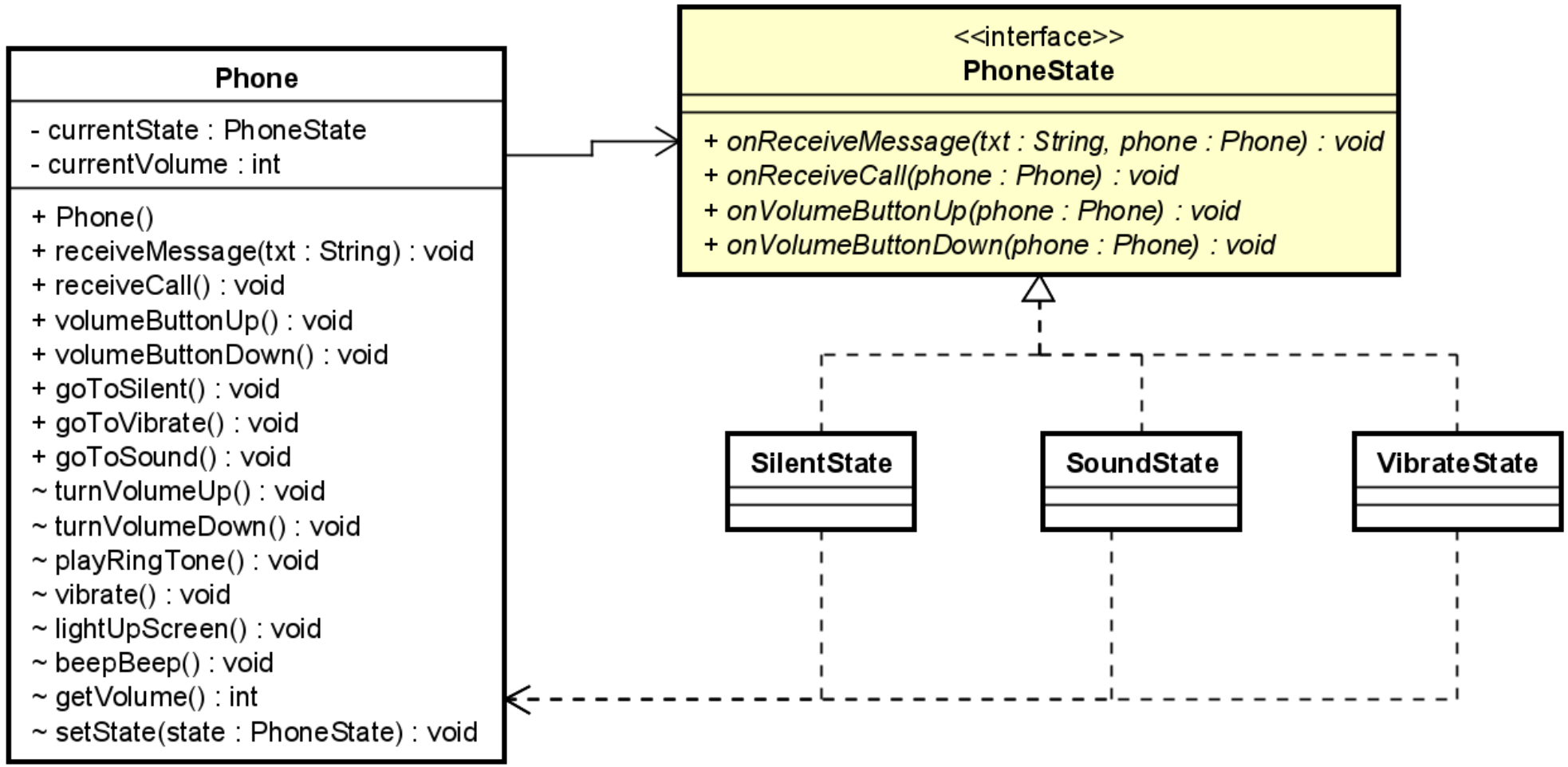

Context
+ method() : void
<<interface>>
State
+ method(ctx : Context) : void
StateImpl1
StateImpl2

Agenda

- Concept by example
  - Automatic doors
  - Radiator
  - Mobile phone
  - Animation/controls in games
  - Threads
- Naïve approach
- State pattern
  - What is the purpose?
  - UML structure
- How to apply the state pattern?


Program
Context
+ method() : void
<<interface>>
State
+ method(ctx : Context) : void
StateImpl1
StateImpl2

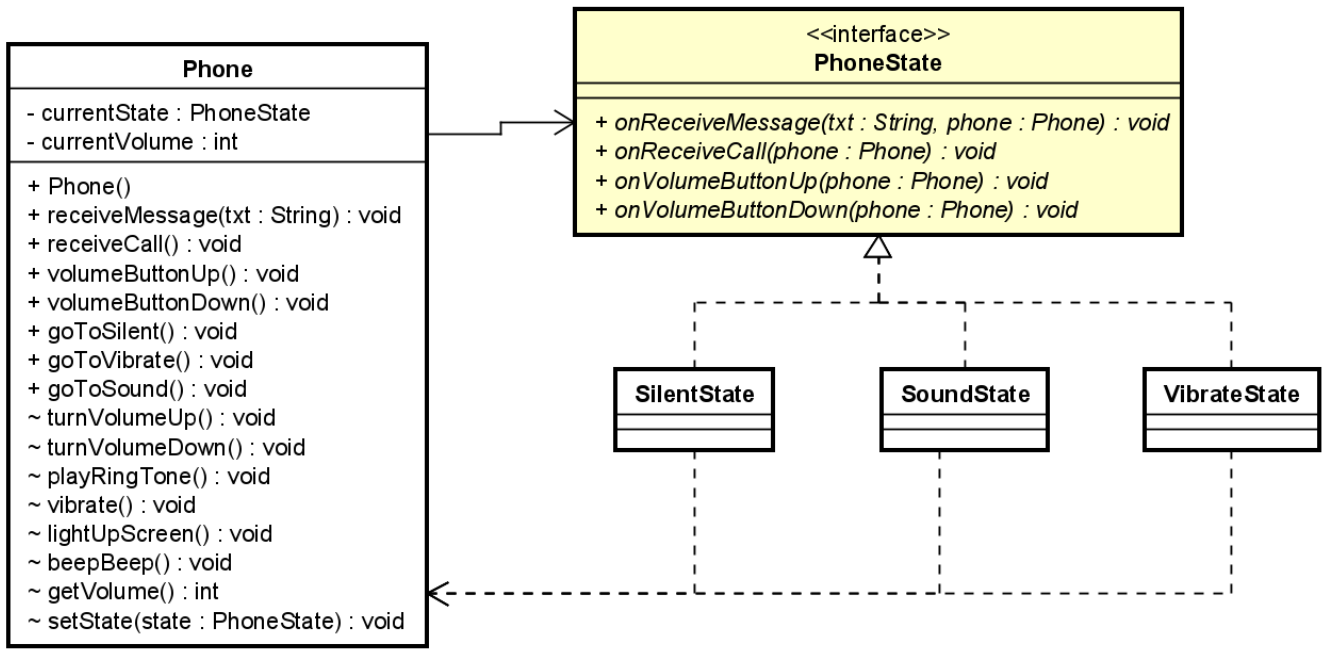

Context
+ method() : void
<<interface>>
State
+ method(ctx : Context) : void
StateImpl1
StateImpl2


Phone
- currentState : PhoneState
- currentVolume : int
+ Phone()
+ receiveMessage(txt : String) : void
+ receiveCall() : void
+ volumeButtonUp() : void
+ volumeButtonDown() : void
+ goToSilent() : void
+ goToVibrate() : void
+ goToSound() : void
~ turnVolumeUp() : void
~ turnVolumeDown() : void
~ playRingTone() : void
~ vibrate() : void
~ lightUpScreen() : void
~ beepBeep() : void
~ getVolume() : int
~ setState(state : PhoneState) : void
<<interface>>
PhoneState
+ onReceiveMessage(txt : String, phone : Phone) : void
+ onReceiveCall(phone : Phone) : void
+ onVolumeButtonUp(phone : Phone) : void
+ onVolumeButtonDown(phone : Phone) : void
SilentState
SoundState
VibrateState

```
public void receiveMessage(String txt) {
        if(isOnSilent) {
            // nothing
        } else if(isOnSound) {
            beepBeep();
        } else if(isOnVibrate) {
            vibrate();
        }
        System.out.println("Message
                                received:");
        System.out.println(txt);
    }
```

```
private PhoneState currentState;

public void receiveMessage(String txt) {
    currentState.onReceiveMessage(txt, this);
}
```



```
    private PhoneState currentState =
                            new SoundState();
    private int currentVolume;

    public void receiveMessage(String txt) {
        currentState.onReceiveMessage(txt, this);
    }

    public void receiveCall() {
        currentState.onReceiveCall(this);
    }

    public void volumeUpButton() {
        currentState.onVolumeButtonUp(this);
    }

    public void goToSilentState() {
        currentState = new SilentState();
    }

    void turnVolumeUp() {
        currentVolume++;
    }

void setState(PhoneState state) {
        currentState = state;
    }

    void playRingTone() {
        System.out.println("Ringeling Ringeling");
    }
    void vibrate() {
```

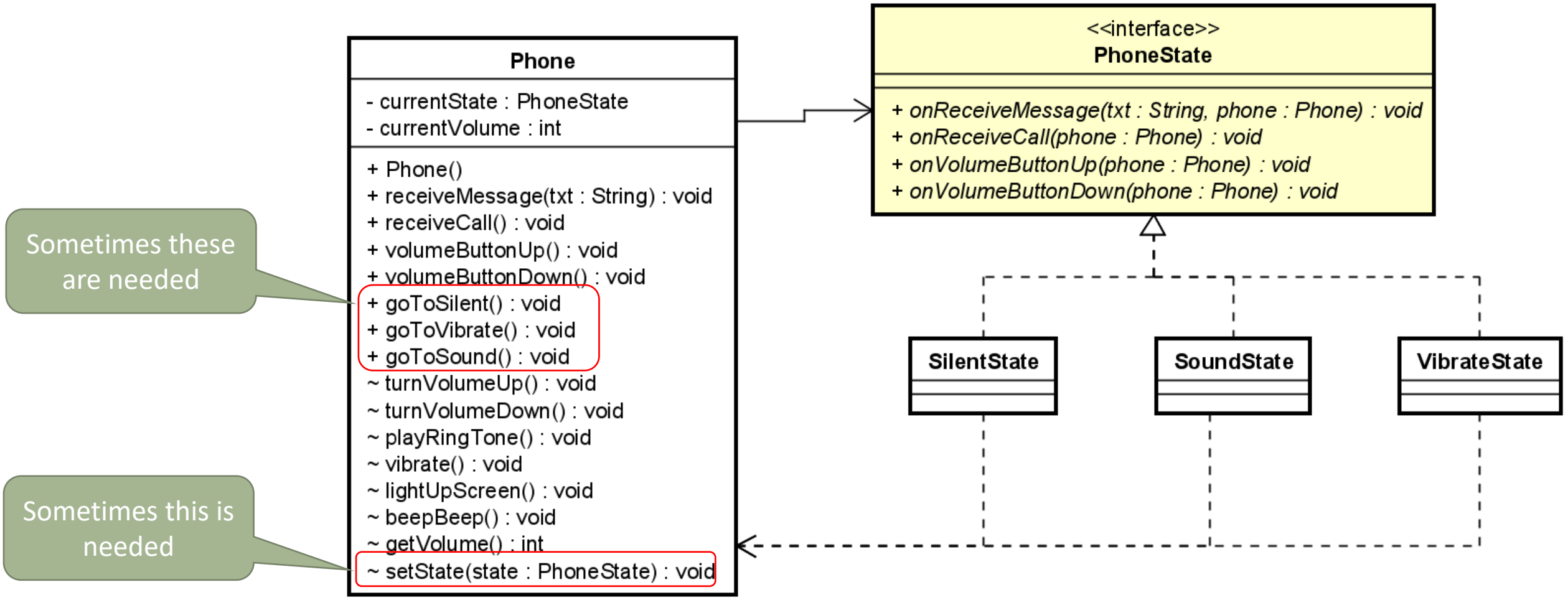
```
public class SoundState implements PhoneState {
    @Override
    public void onReceiveMessage(String txt, Phone phone) {
        phone.beepBeep();
        System.out.println(txt);
    }

    @Override
    public void onReceiveCall(Phone phone) {
        phone.playRingTone();
    }

    @Override
    public void onVolumeButtonUp(Phone phone) {
        int vol = phone.getVolume();
        if(vol < 100) {
            phone.turnVolumeUp();
        }
    }

    @Override
    public void onVolumeButtonDown(Phone phone) {
        int vol = phone.getVolume();
        if(vol > 1) {
            phone.turnVolumeDown();
        } else {
            phone.setState(new SilentState());
        }
    }
}
```

Sometimes the state can be changed from “the outside”, sometimes from “the inside”



Agenda

- Concept by example
  - Automatic doors
  - Radiator
  - Mobile phone
  - Animation/controls in games
  - Threads
- Naïve approach
- State pattern
  - What is the purpose?
  - UML structure
- How to apply the state pattern?

Applicability

- When you have an object that behaves differently depending on its current state.
- Large number of states
- When a class is polluted with massive conditionals that alter methods’s behavior according to the current values of the class's fields.

⚖ Pros and Cons

- ✔ Eliminates state machine conditionals.
- ✔ Organizes the code related to particular states into separate classes.
- ✔ Simplifies the code of the context.

- ✖ Can be an overkill if a state machine has only a few states or rarely changes.
- ✖ Duplicate code between states
- ✖ If you’re not careful with your interface, some states may just have empty method implementations, which is usually not good.