State Pattern

Jumper Example

State Pattern Example

- Simulate a TV without using control flow (ie. Use the state pattern)
- Create a Class named TV with no constructor parameters
- The TV must contain the following methods as its API:
  - volumeUp(): Unit
  - volumeDown(): Unit
  - mute(): Unit
  - power(): Unit
  - currentVolume(): Int

TV Spec Sheet

- TV is initially off when created
- Initial volume is 5
- When the TV is off:
  - Volume up/down and mute buttons do nothing
  - Current volume is 0
- The power button turns the TV on/off
- Volume up button increases volume by 1 up to a maximum volume of 10
- Volume down button decreases volume by 1 down to minimum volume of 0
- Pressing the mute button mutes/unmutes the TV
- When the TV is muted:
  - Current volume is 0
  - Pressing the mute, volume up, or volume down buttons will unmute the TV and restore the volume to the pre-mute volume (Do not in/decrease the volume)
- When turning the TV back on, the volume should return to its value when the TV was last on
- If the TV was turned off while muted, when it is turned back on it should not be muted

State Pattern Example

We could write all this behavior without the state pattern

- But we're here for state pattern practice so lets use it

TV Spec Sheet

- TV is initially off when created
- Initial volume is 5
- When the TV is off:
  - Volume up/down and mute buttons do nothing
  - Current volume is 0
- The power button turns the TV on/off
- Volume up button increases volume by 1 up to a maximum volume of 10
- Volume down button decreases volume by 1 down to minimum volume of 0
- Pressing the mute button mutes/unmutes the TV
- When the TV is muted:
  - Current volume is 0
  - Pressing the mute, volume up, or volume down buttons will unmute the TV and restore the volume to the pre-mute volume (Do not in/decrease the volume)
- When turning the TV back on, the volume should return to its value when the TV was last on
- If the TV was turned off while muted, when it is turned back on it should not be muted

State Pattern Example

How to implement these features?

- Write your API
  - What methods will change behavior depending on the current state of the object
  - These methods define your API and are declared in the state abstract class
- Decide what states should exist
  - Any situation where the behavior is different should be a new state
- Determine the transitions between states

TV Spec Sheet

- TV is initially off when created
- Initial volume is 5
- When the TV is off:
  - Volume up/down and mute buttons do nothing
  - Current volume is 0
- The power button turns the TV on/off
- Volume up button increases volume by 1 up to a maximum volume of 10
- Volume down button decreases volume by 1 down to minimum volume of 0
- Pressing the mute button mutes/unmutes the TV
- When the TV is muted:
  - Current volume is 0
  - Pressing the mute, volume up, or volume down buttons will unmute the TV and restore the volume to the pre-mute volume (Do not in/decrease the volume)
- When turning the TV back on, the volume should return to its value when the TV was last on
- If the TV was turned off while muted, when it is turned back on it should not be muted

State Pattern Example

How to implement these features?

- Write your API
  - What methods will change behavior depending on the current state of the object

API:

- This API contains methods for all the buttons on the TV, and a method to get the current volume
  - volumeUp()
  - volumeDown()
  - mute()
  - power()
  - currentVolume()

TV Spec Sheet

- TV is initially off when created
- Initial **volume** is 5
- When the TV is off:
  - **Volume up/down** and **mute buttons** do nothing
  - Current **volume** is 0
- The **power button** turns the TV on/off
- **Volume up button** increases volume by 1 up to a maximum volume of 10
- **Volume down button** decreases volume by 1 down to minimum volume of 0
- Pressing the **mute button** mutes/unmutes the TV
- When the TV is muted:
  - Current **volume** is 0
  - Pressing the **mute**, **volume up**, or **volume down buttons** will unmute the TV and restore the **volume** to the pre-mute **volume** (Do not in/decrease the volume)
- When turning the TV back **on**, the **volume** should return to its value when the TV was last on
- If the TV was **turned off** while muted, when it is **turned back on** it should not be muted

State Pattern Example

How to implement these features?

- Decide what states should exist

States:

TV Spec Sheet

- TV is initially off when created
- Initial volume is 5
- When the TV is off:
  - Volume up/down and mute buttons do nothing
  - Current volume is 0
- The power button turns the TV on/off
- Volume up button increases volume by 1 up to a maximum volume of 10
- Volume down button decreases volume by 1 down to minimum volume of 0
- Pressing the mute button mutes/unmutes the TV
- When the TV is muted:
  - Current volume is 0
  - Pressing the mute, volume up, or volume down buttons will unmute the TV and restore the volume to the pre-mute volume (Do not in/decrease the volume)
- When turning the TV back on, the volume should return to its value when the TV was last on
- If the TV was turned off while muted, when it is turned back on it should not be muted

State Pattern Example

How to implement these features?

- Decide what states should exist

States:

- Off <-- Initial State
- On (but not muted)
- Muted

TV Spec Sheet

- TV is **initially off** when created
- Initial volume is 5
- When the TV is **off**:
  - Volume up/down and mute buttons do nothing
  - Current volume is 0
- The power button turns the TV on/off
- Volume up button increases volume by 1 up to a maximum volume of 10
- Volume down button decreases volume by 1 down to minimum volume of 0
- Pressing the mute button **mutes/unmutes** the TV
- When the TV is **muted**:
  - Current volume is 0
  - Pressing the mute, volume up, or volume down buttons will unmute the TV and restore the volume to the pre-mute volume (Do not in/decrease the volume)
- When turning the TV back **on**, the volume should return to its value when the TV was last **on**
- If the TV was turned **off** while muted, when it is turned back **on** it should **not be muted**

State Pattern Example

How to implement these features?

- Determine the transitions between states

State Transitions:

TV Spec Sheet

- TV is initially off when created
- Initial volume is 5
- When the TV is off:
  - Volume up/down and mute buttons do nothing
  - Current volume is 0
- The power button turns the TV on/off
- Volume up button increases volume by 1 up to a maximum volume of 10
- Volume down button decreases volume by 1 down to minimum volume of 0
- Pressing the mute button mutes/unmutes the TV
- When the TV is muted:
  - Current volume is 0
  - Pressing the mute, volume up, or volume down buttons will unmute the TV and restore the volume to the pre-mute volume (Do not in/decrease the volume)
- When turning the TV back on, the volume should return to its value when the TV was last on
- If the TV was turned off while muted, when it is turned back on it should not be muted

State Pattern Example

How to implement these features?

- Determine the transitions between states

State Transitions:

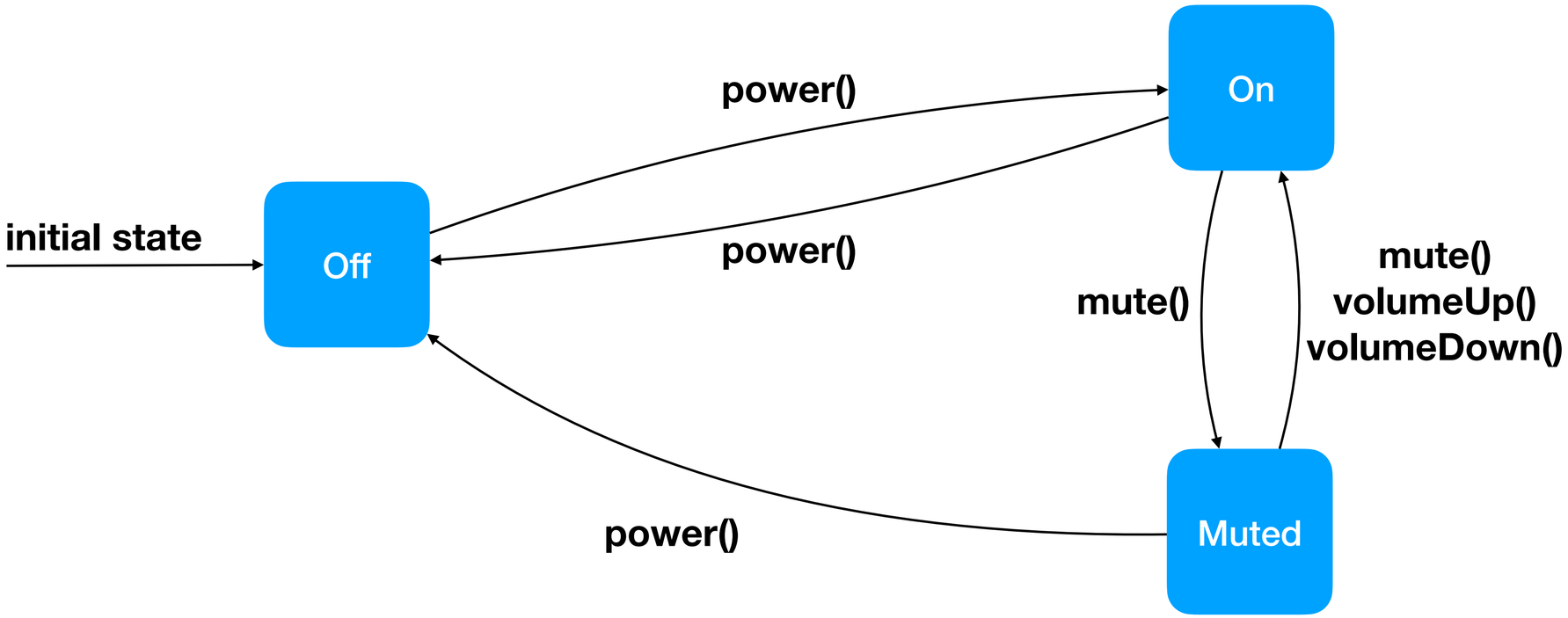
- Off -> On
  - Power button pressed
- On -> Off
  - Power button pressed
- On -> Muted
  - Mute button pressed
- Muted -> On
  - Mute, volume up, or volume down button pressed
- Muted -> Off
  - Power button pressed

TV Spec Sheet

- TV is initially off when created
- Initial volume is 5
- When the TV is off:
  - Volume up/down and mute buttons do nothing
  - Current volume is 0
- **The power button turns the TV on/off**
- Volume up button increases volume by 1 up to a maximum volume of 10
- Volume down button decreases volume by 1 down to minimum volume of 0
- **Pressing the mute button mutes/unmutes the TV**
- When the TV is muted:
  - Current volume is 0
  - **Pressing the mute, volume up, or volume down buttons will unmute the TV** and restore the volume to the pre-mute volume (Do not in/decrease the volume)
- When turning the TV back on, the volume should return to its value when the TV was last on
- If the TV was **turned off while muted**, when it is **turned back on it should not be muted**

State Pattern Example

Let's visualize the states and transitions in a state diagram



```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```

Memory Diagram!!

- This code implements a subset of the required features
  - Full solution in the repo

What happens in memory when this program executes?

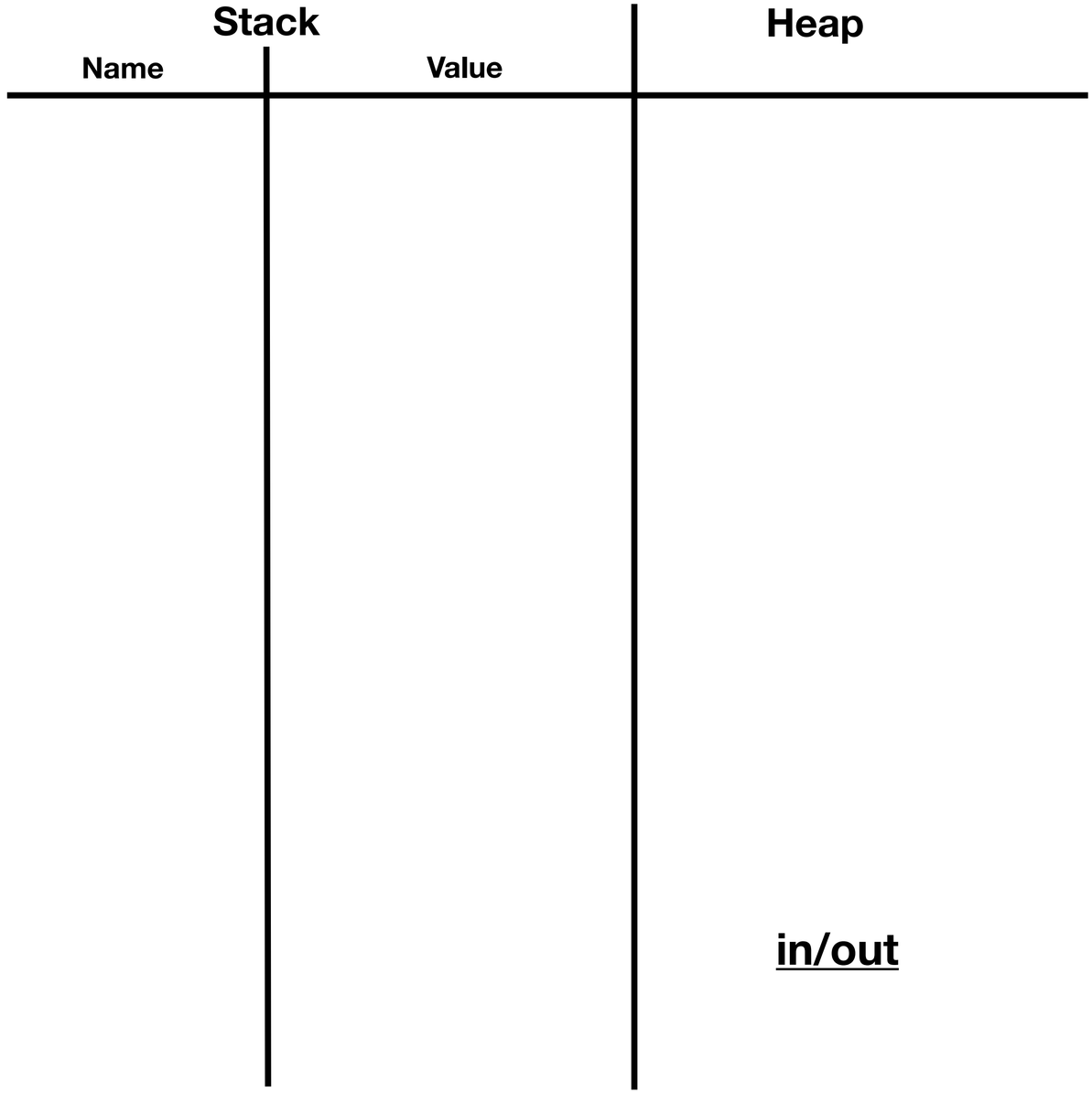
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Let's get started!

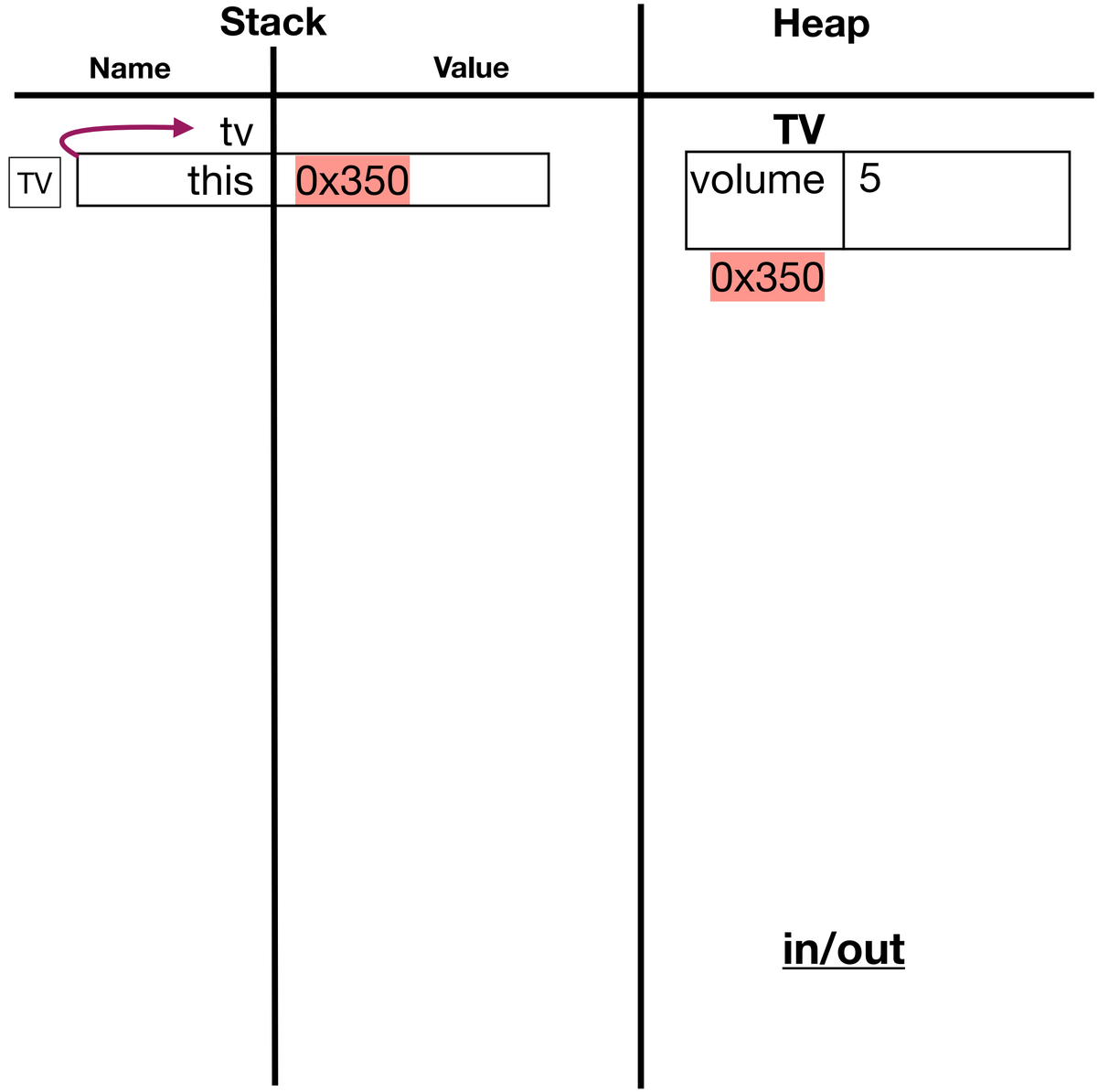
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- I'm not showing the arrows for references anymore
- You should still use arrows on the quiz

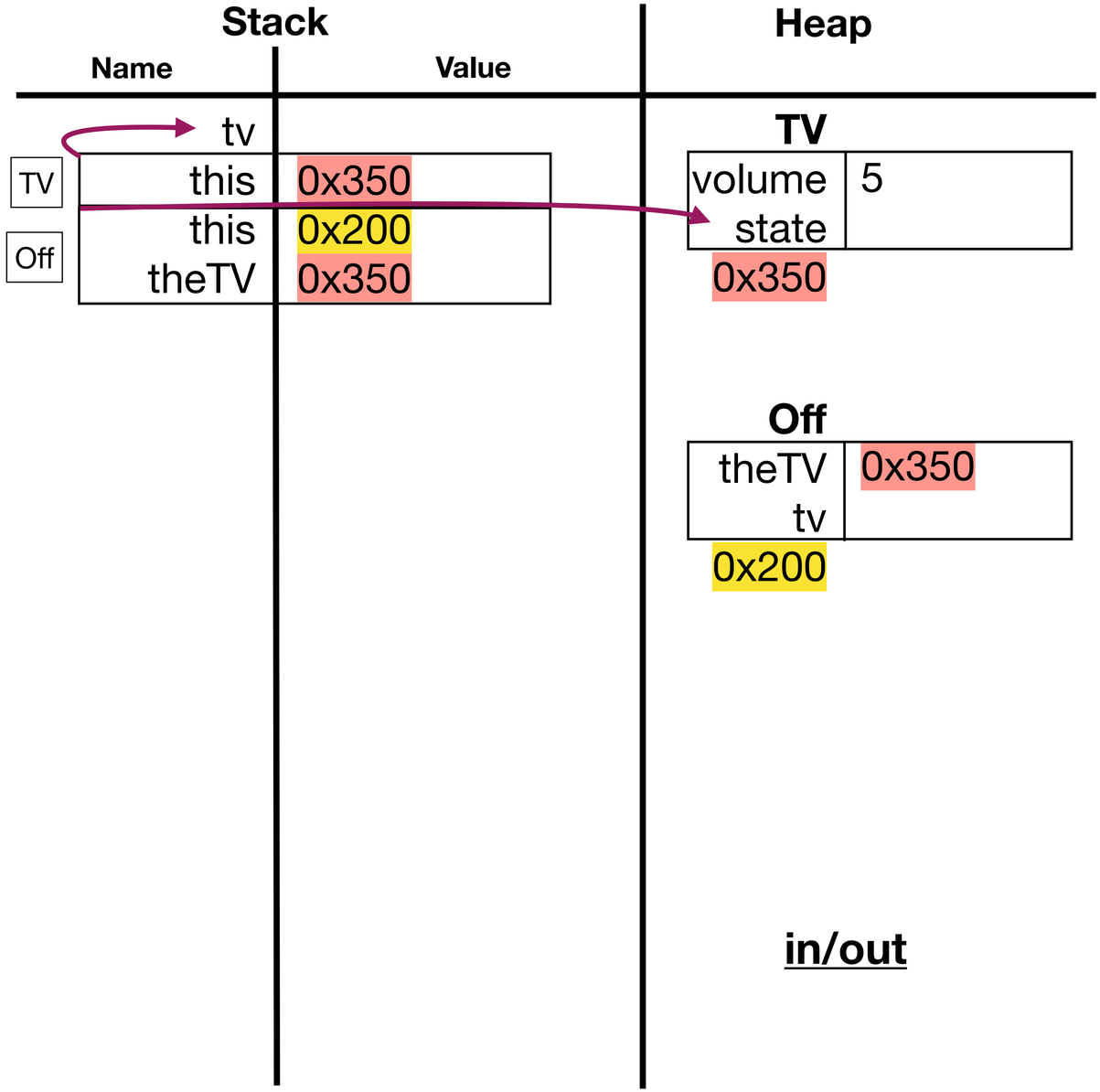
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- The TV constructor creates a Off object
- Calls the Off constructor, but not through inheritance

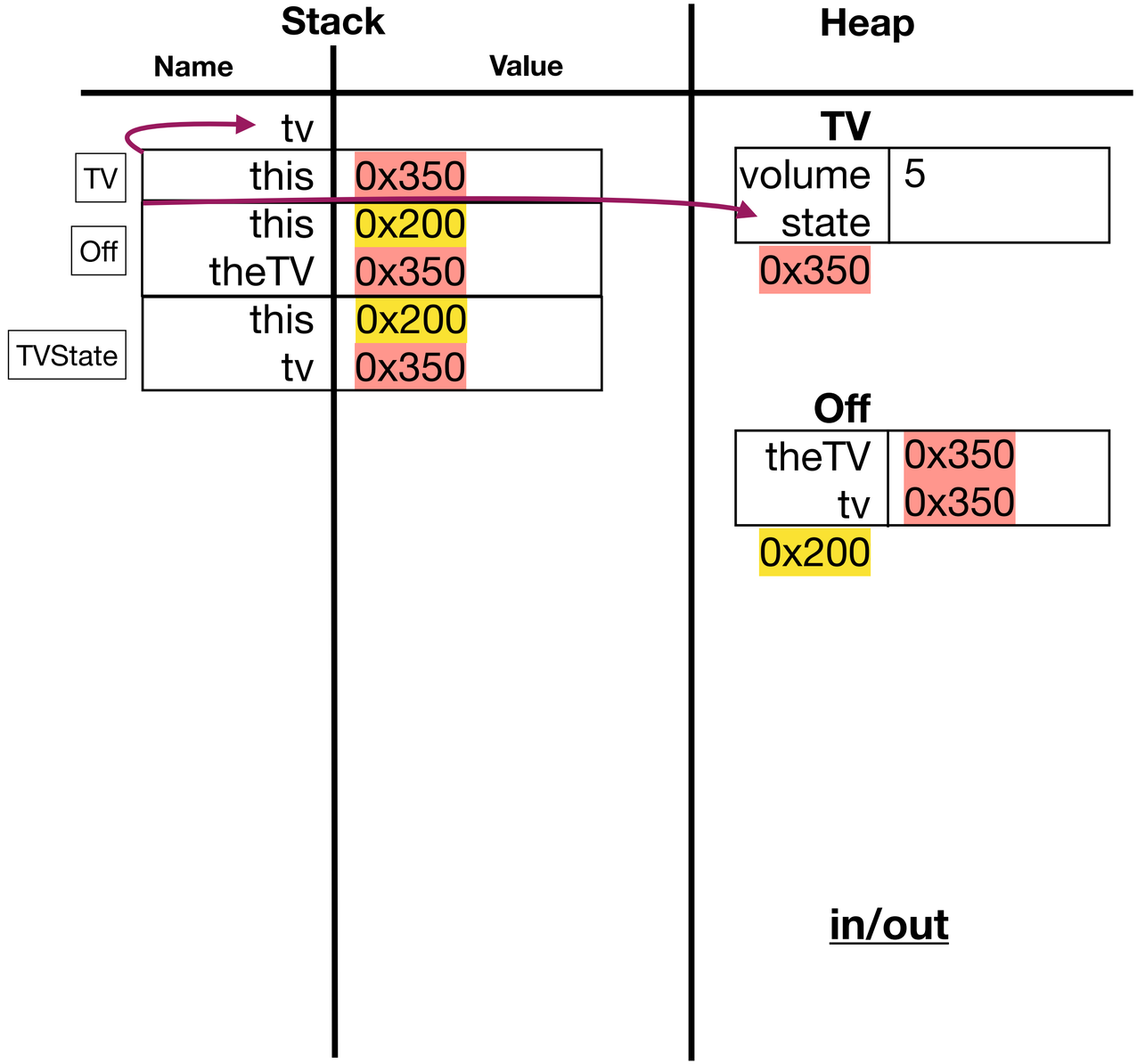
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Off constructor calls the TVState constructor
- This call is due to inheritance

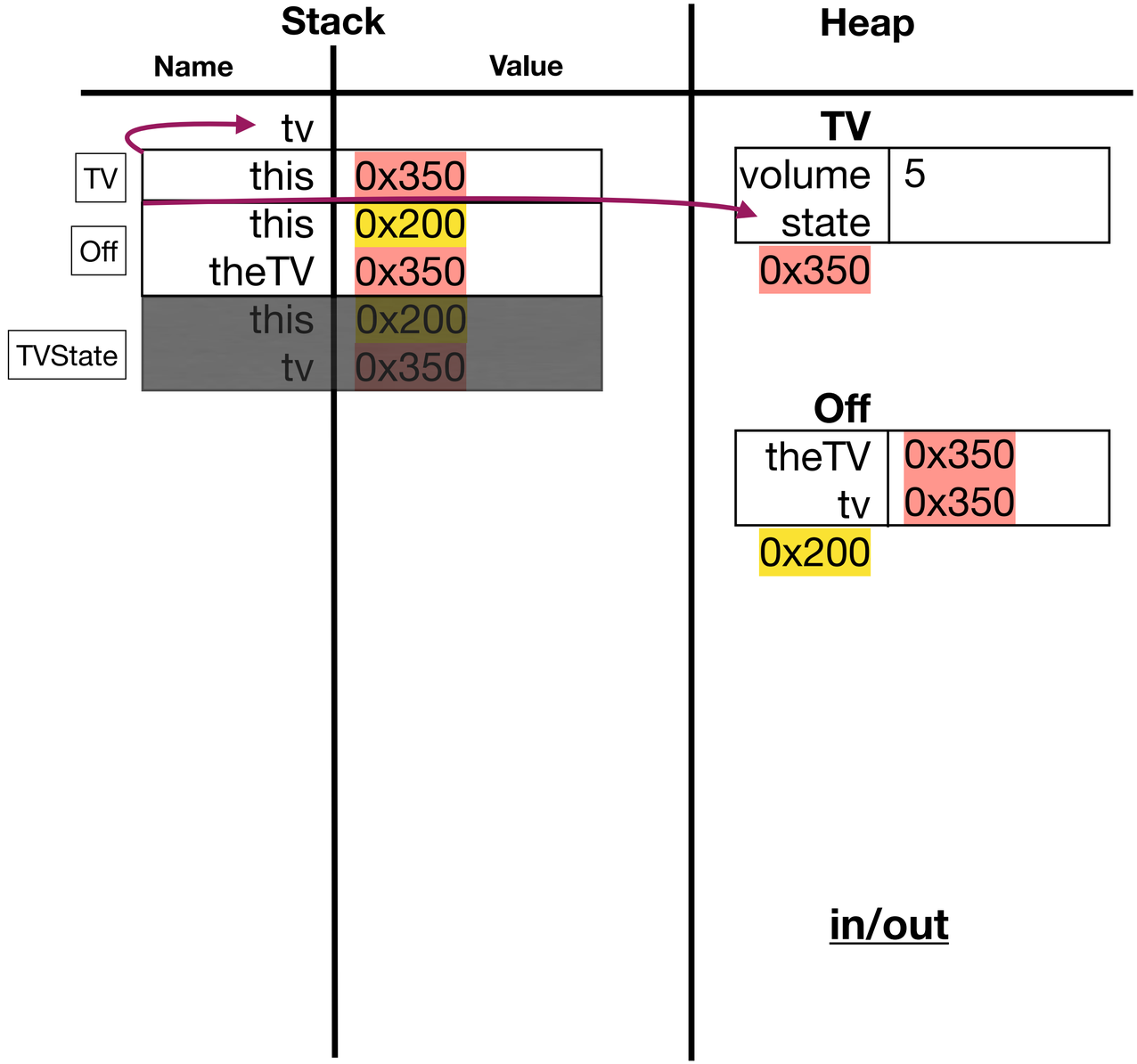
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Notation change:
  - Destroyed stack frames will be greyed out on slides
  - Same as crossing it out, but looks cleaner on a slide

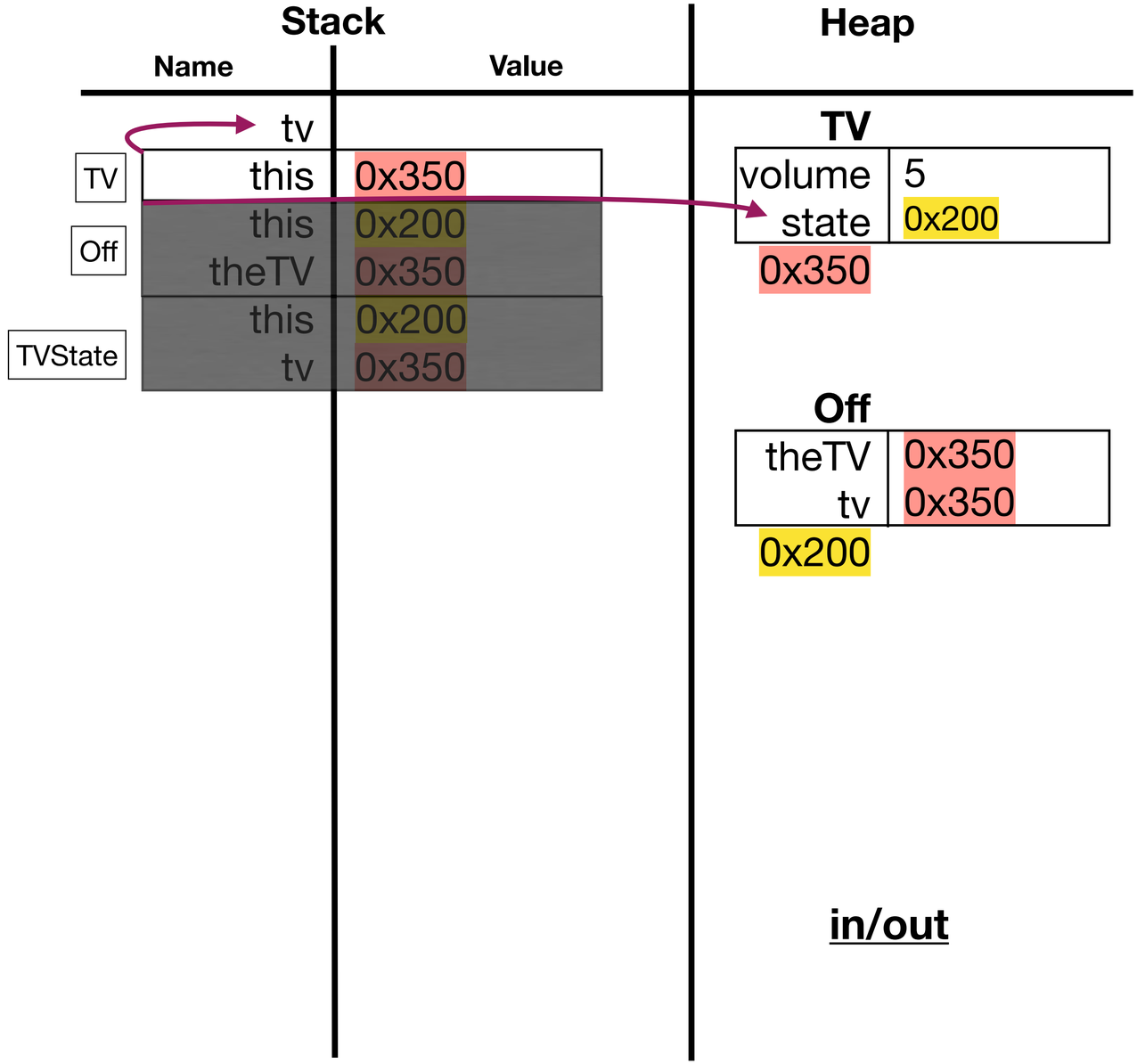
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Off constructor returns a reference to the variable named state

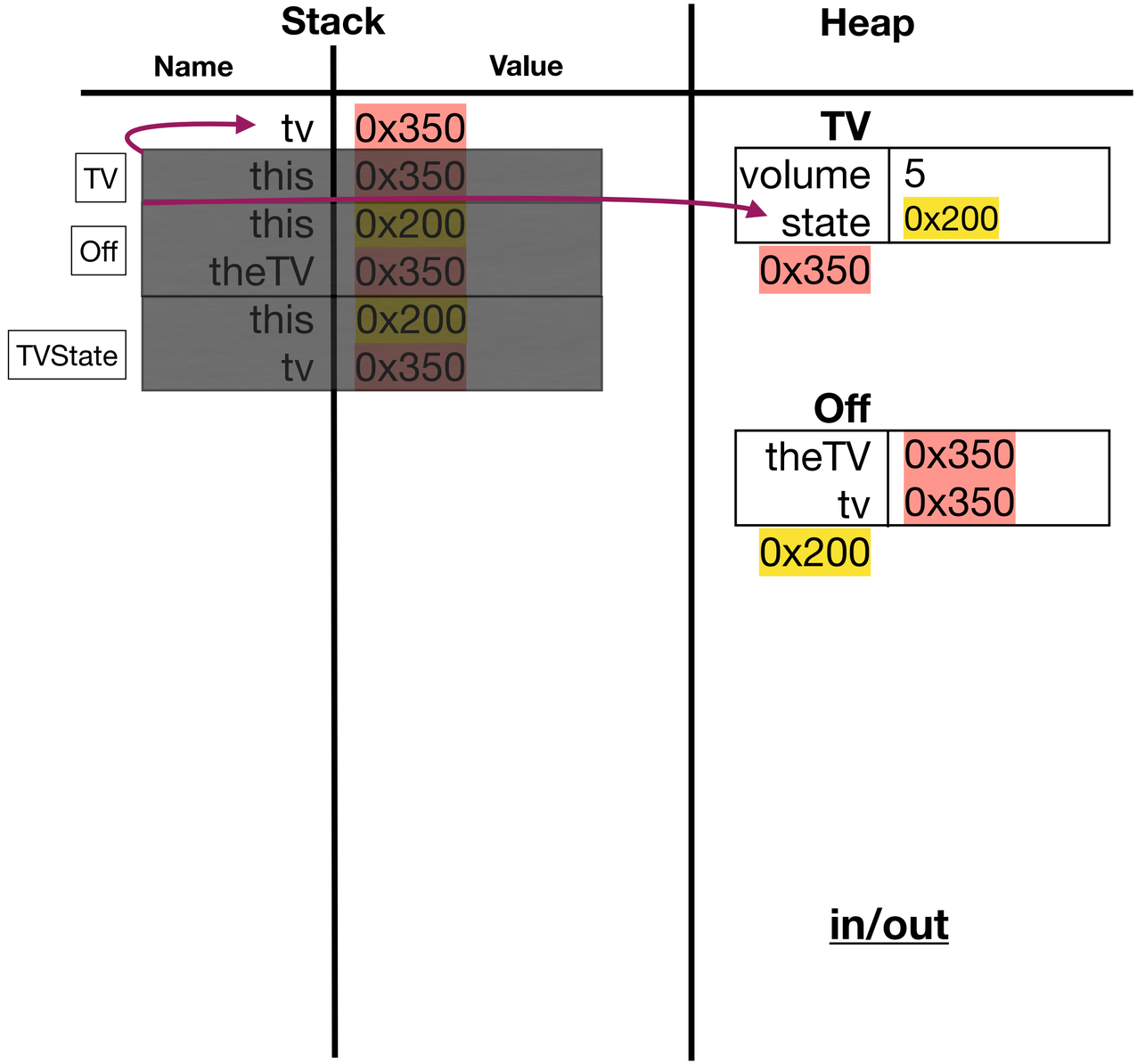
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- TV constructor returns

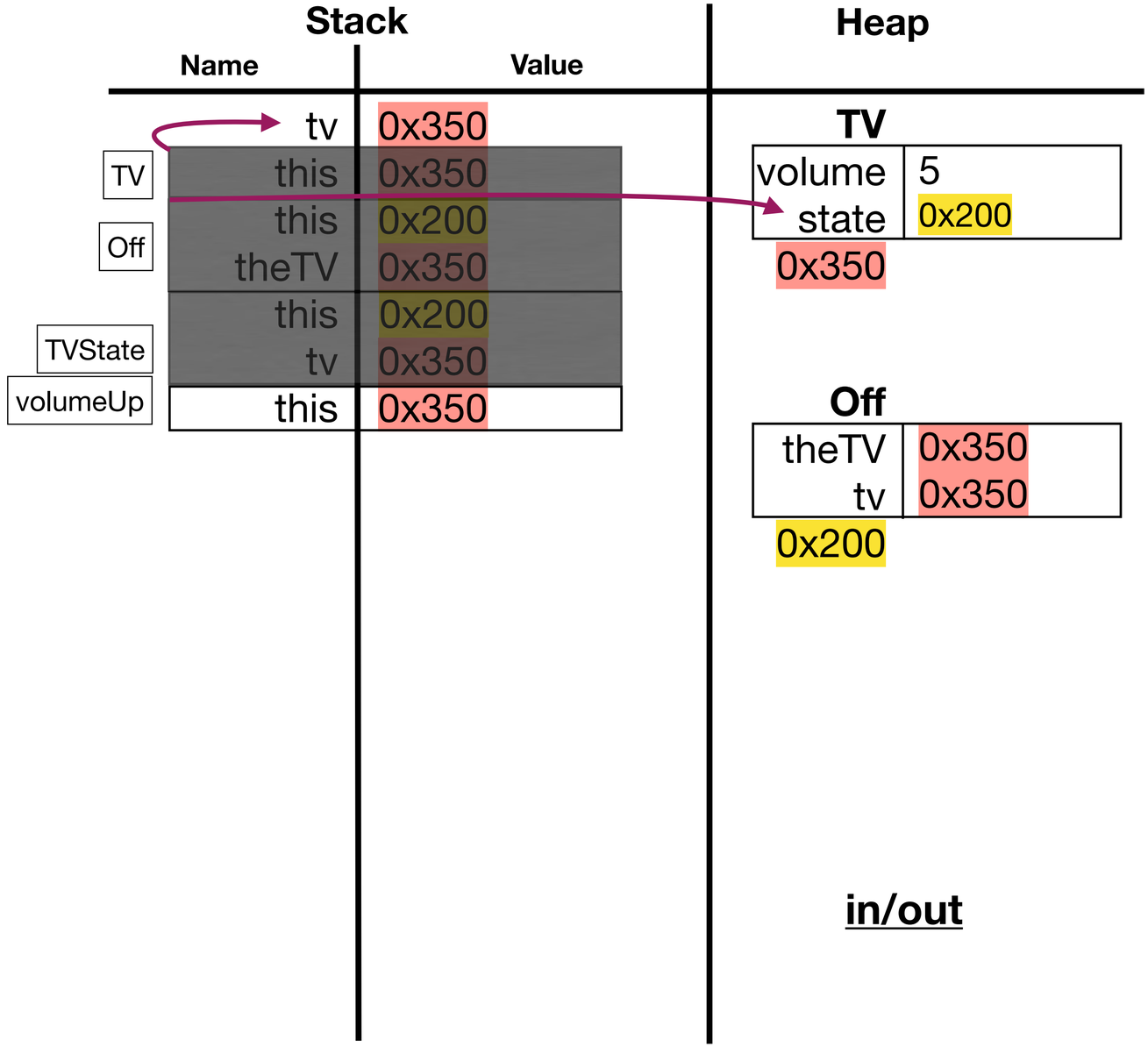
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Call the TV's volumeUp method
- TV defers to it's state for the behavior

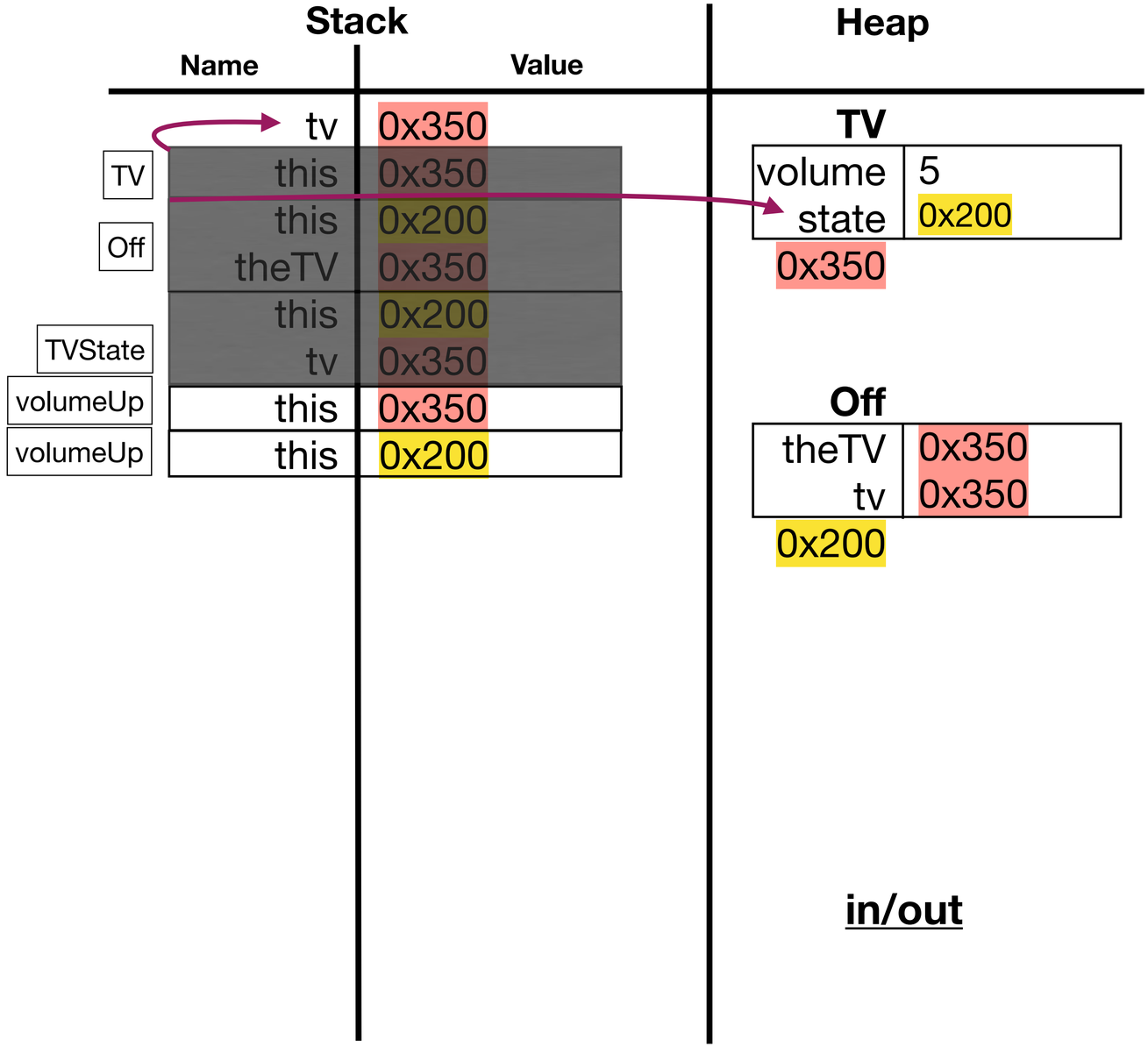
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- The state is currently "Off"
- Off does not override volumeUp; Use TVState's behavior

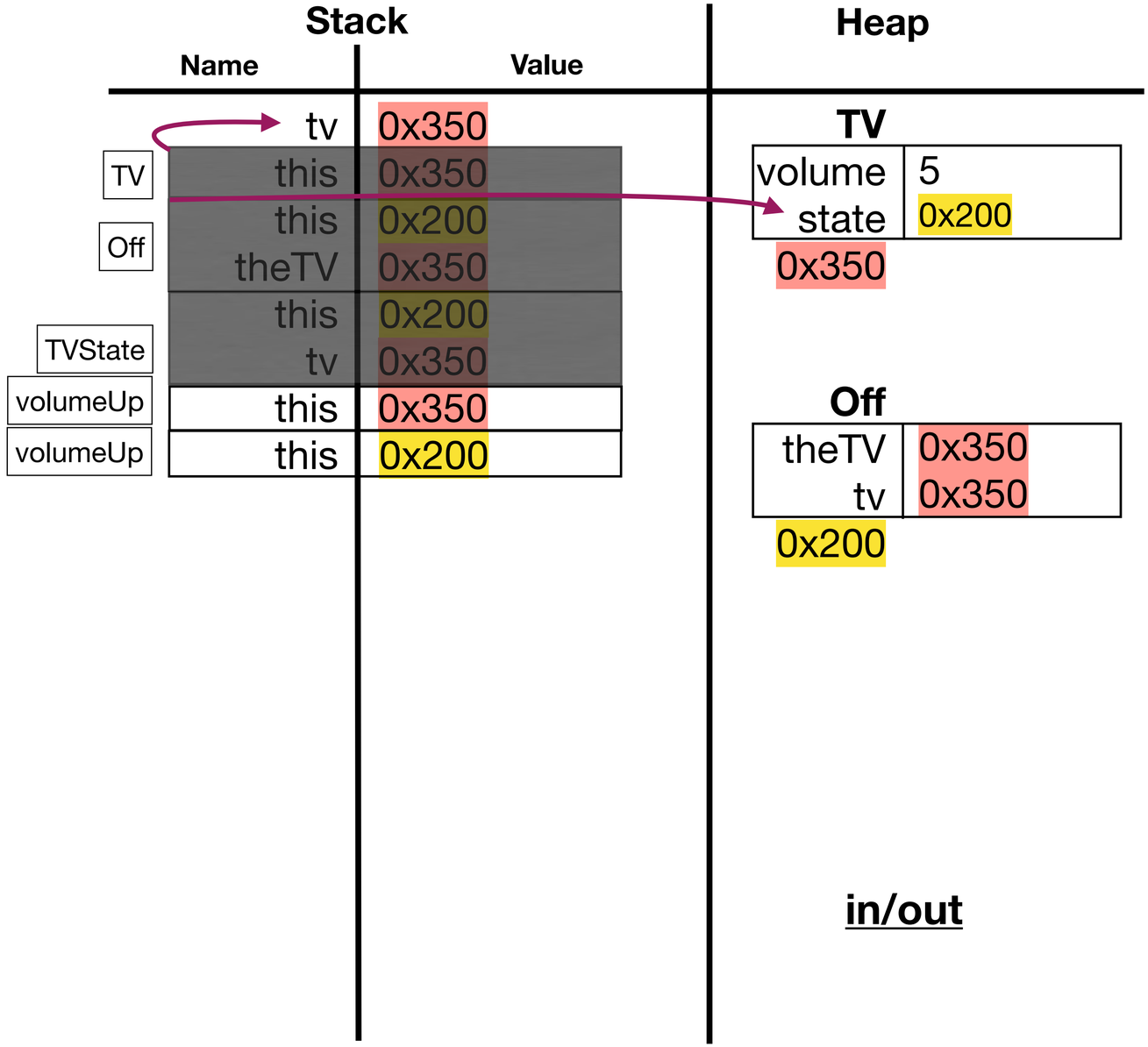
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- The method does nothing
- When the TV is off, the volume up button shouldn't do anything

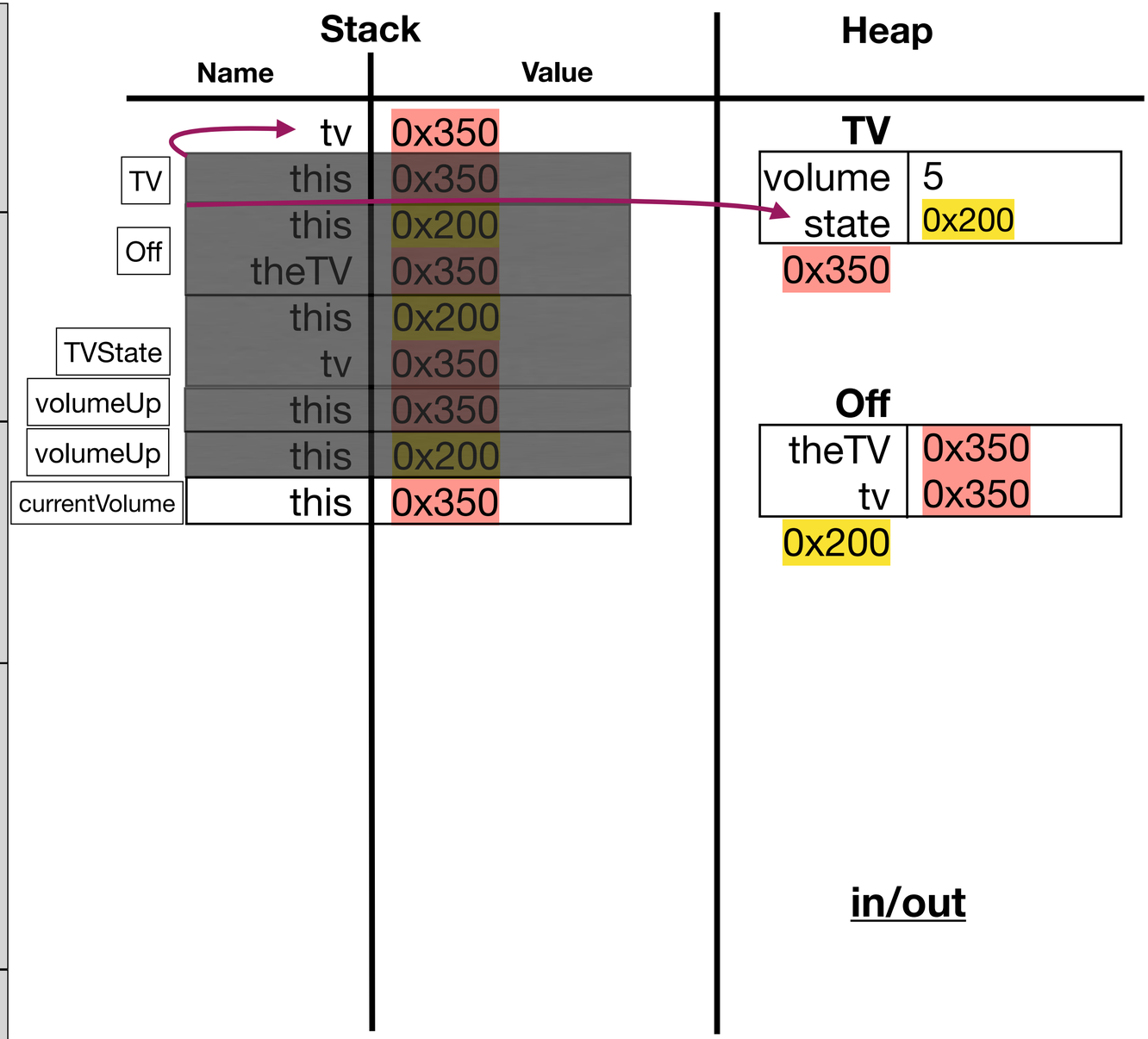
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Same process for currentVolume
- TV defers to it's state for functionality

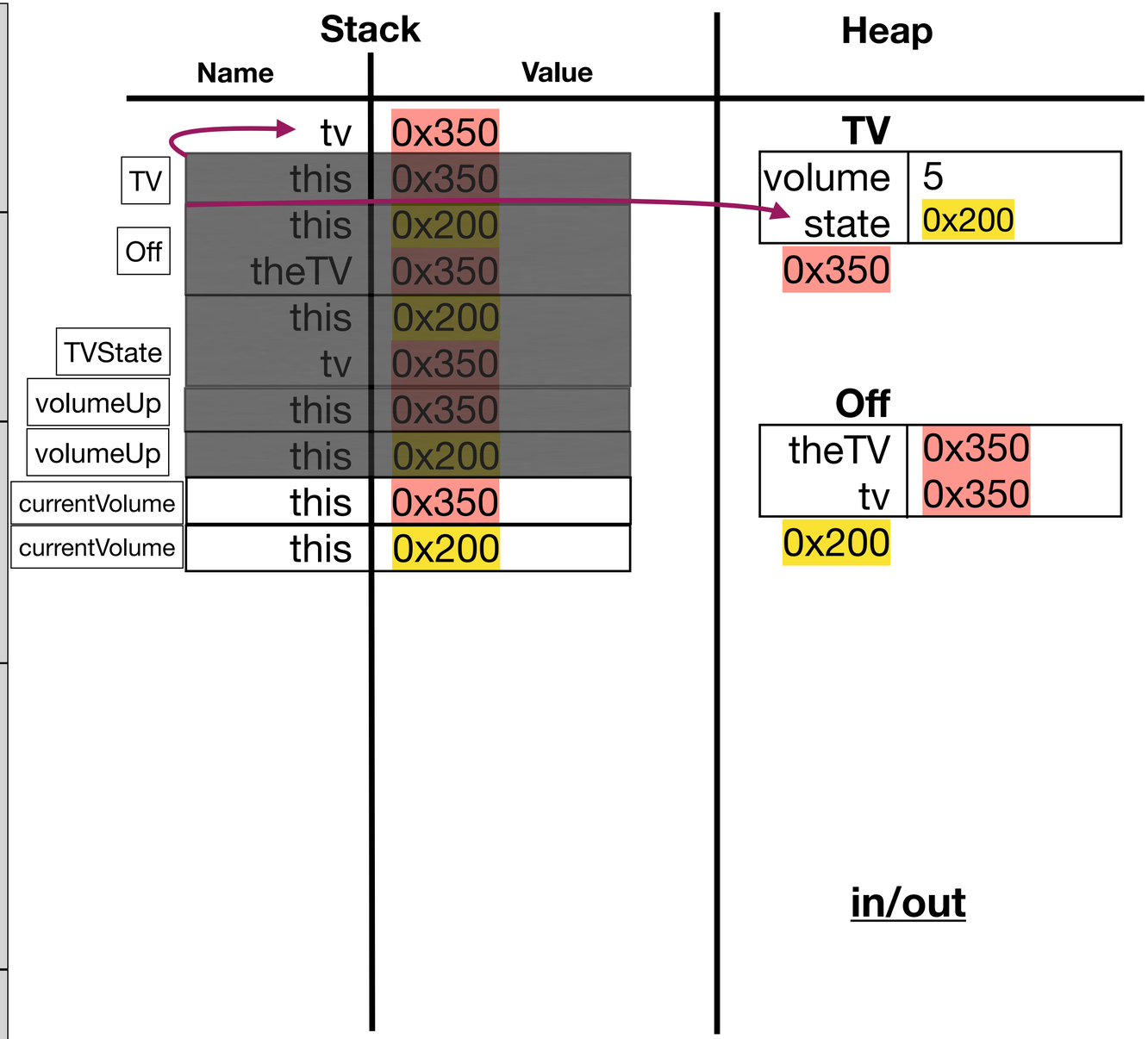
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Off overrides currentVolume to return 0
- This is the behavior we want when the TV is off

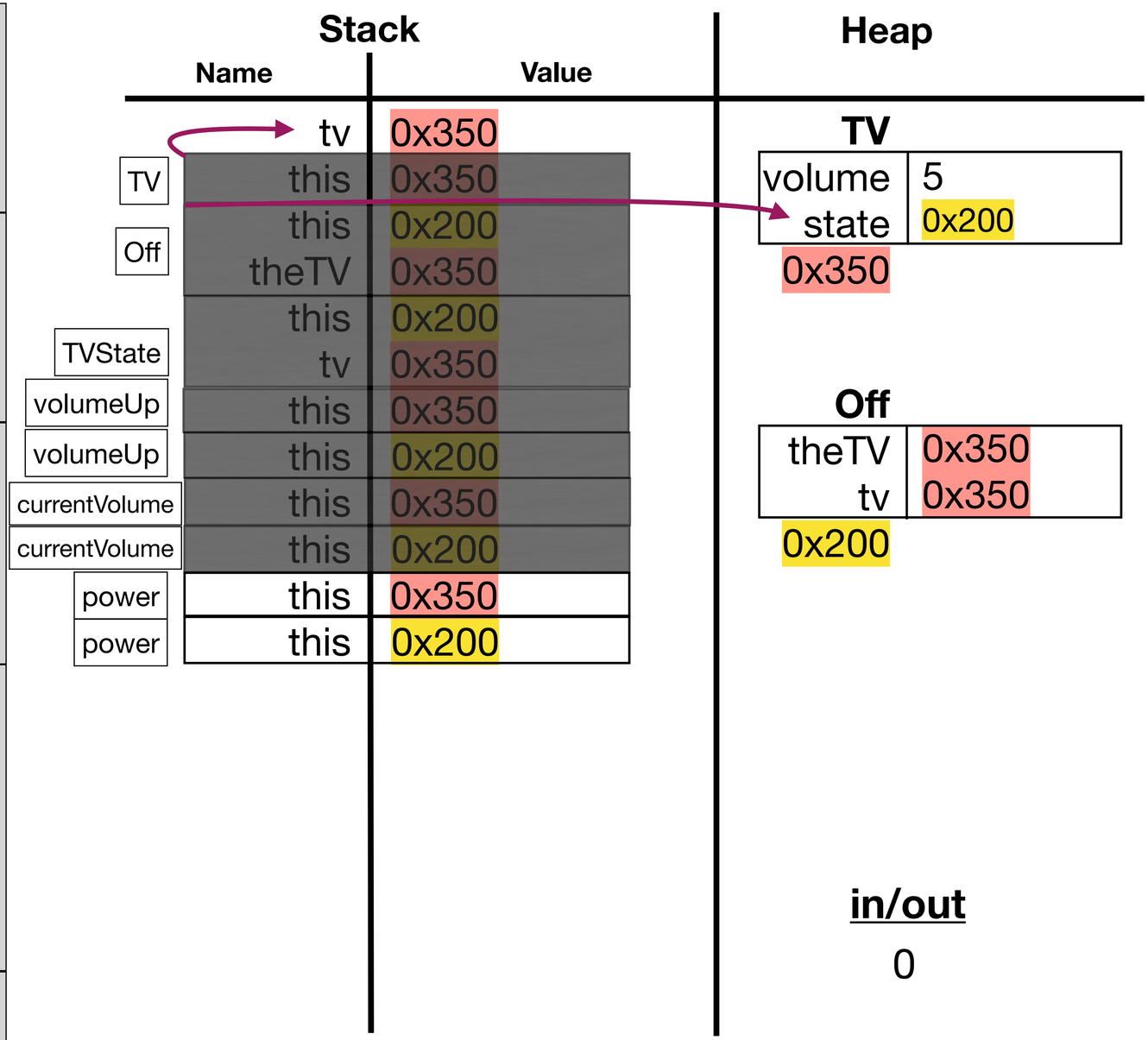
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Off overrides power
- This is our first state transition

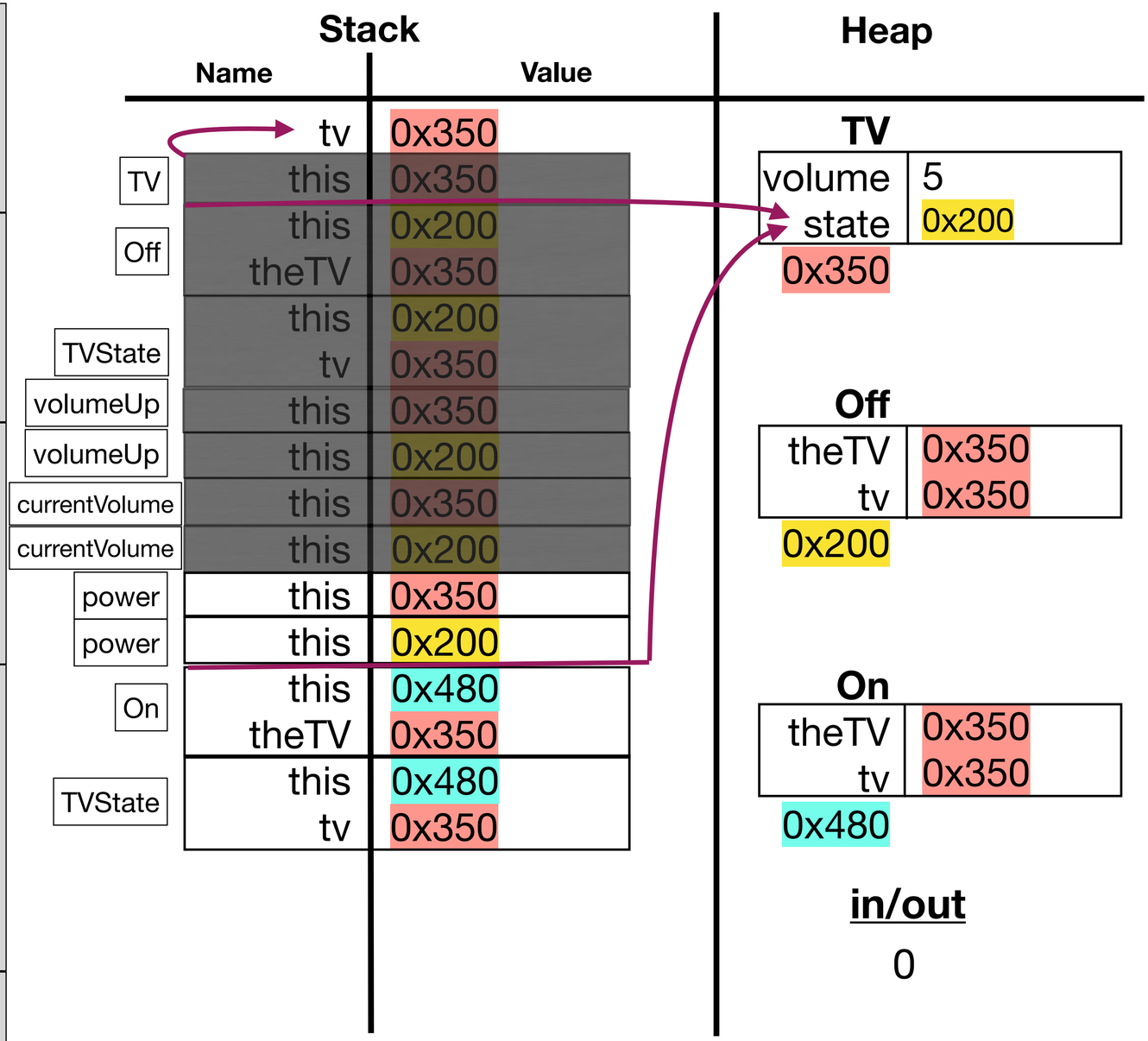
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- A new object of type On is created

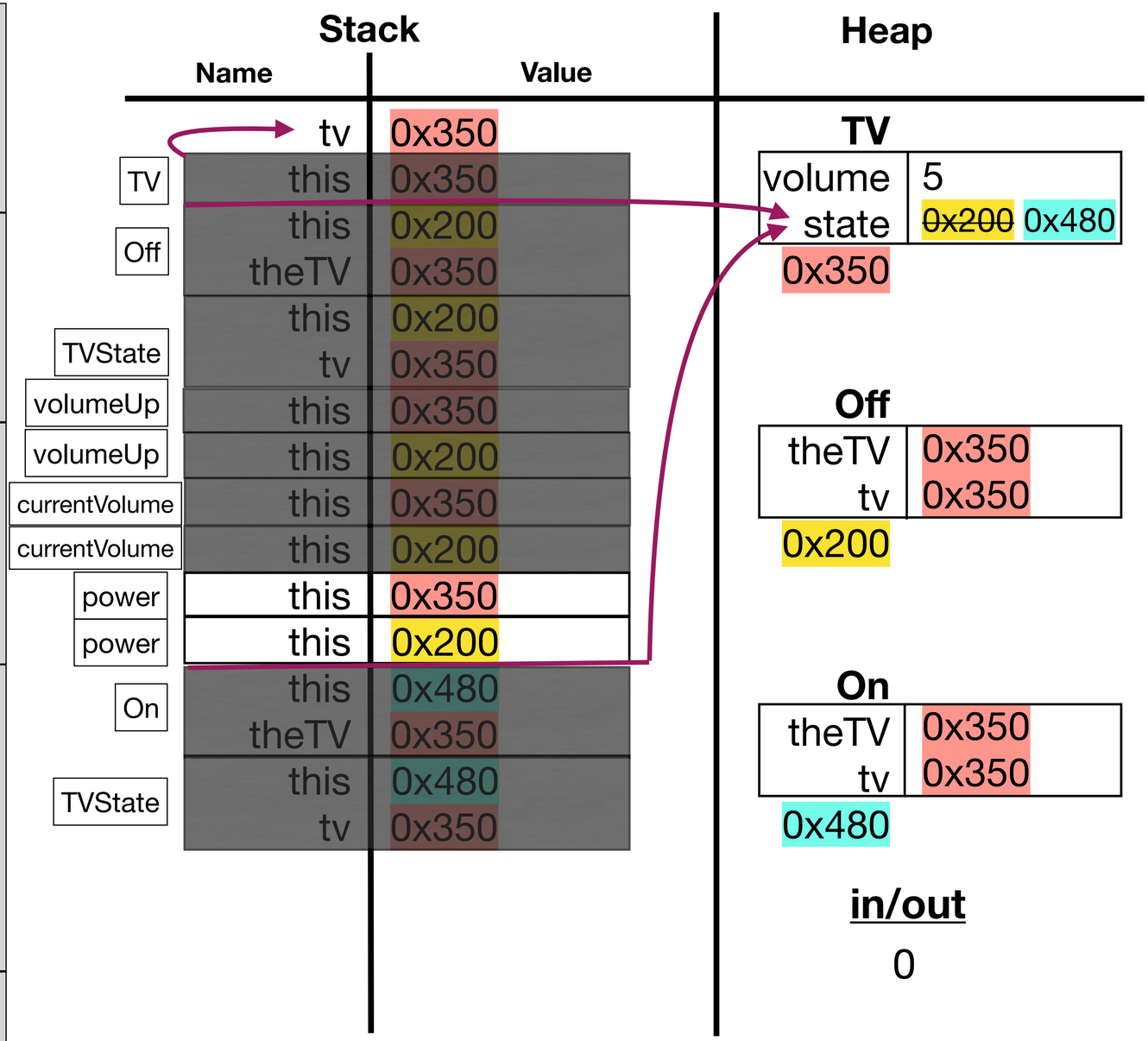
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- The state transition will replace the state of the TV with the new On state

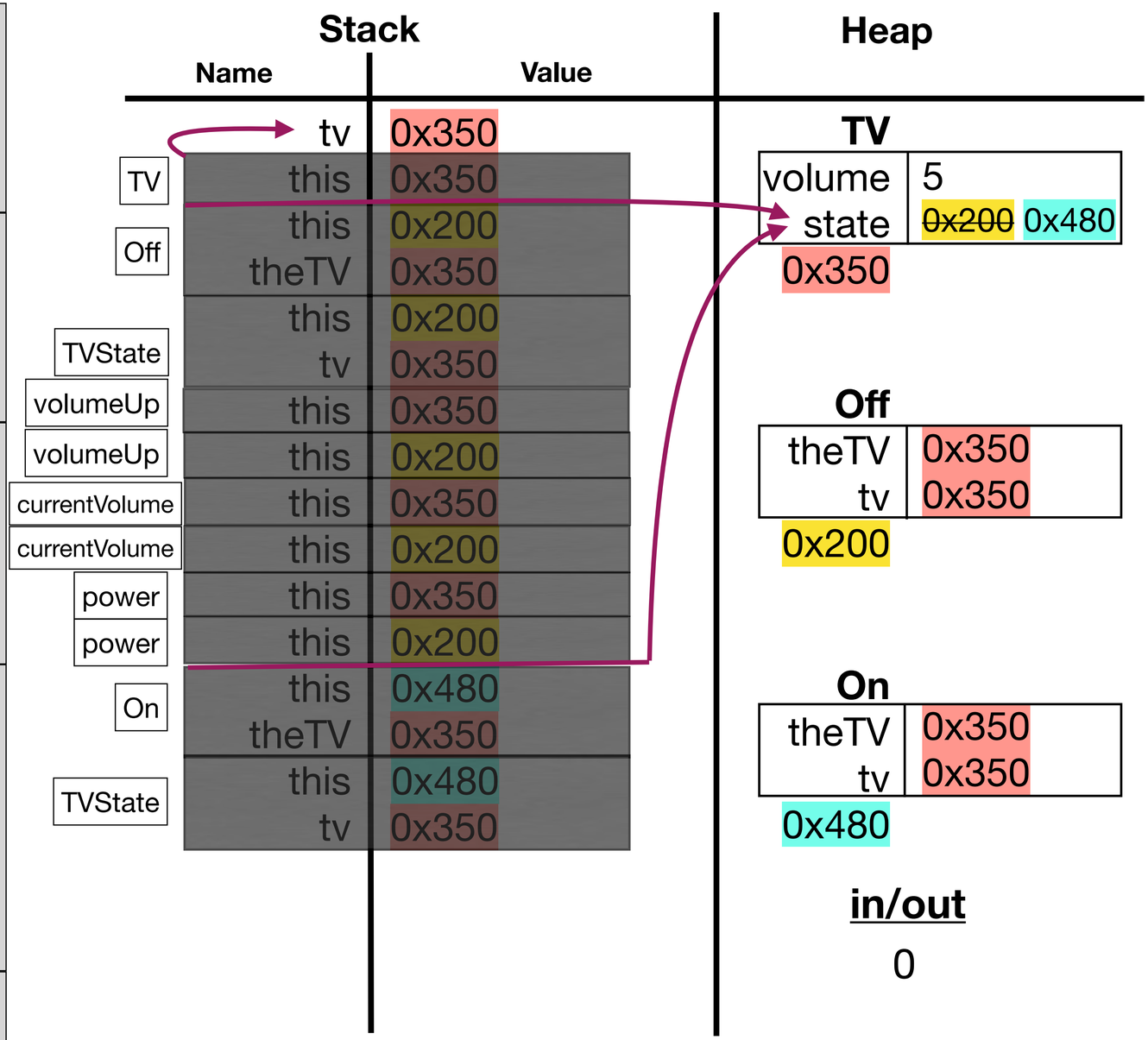
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- After pressing the power button
  - The TV is on and has completely different behavior

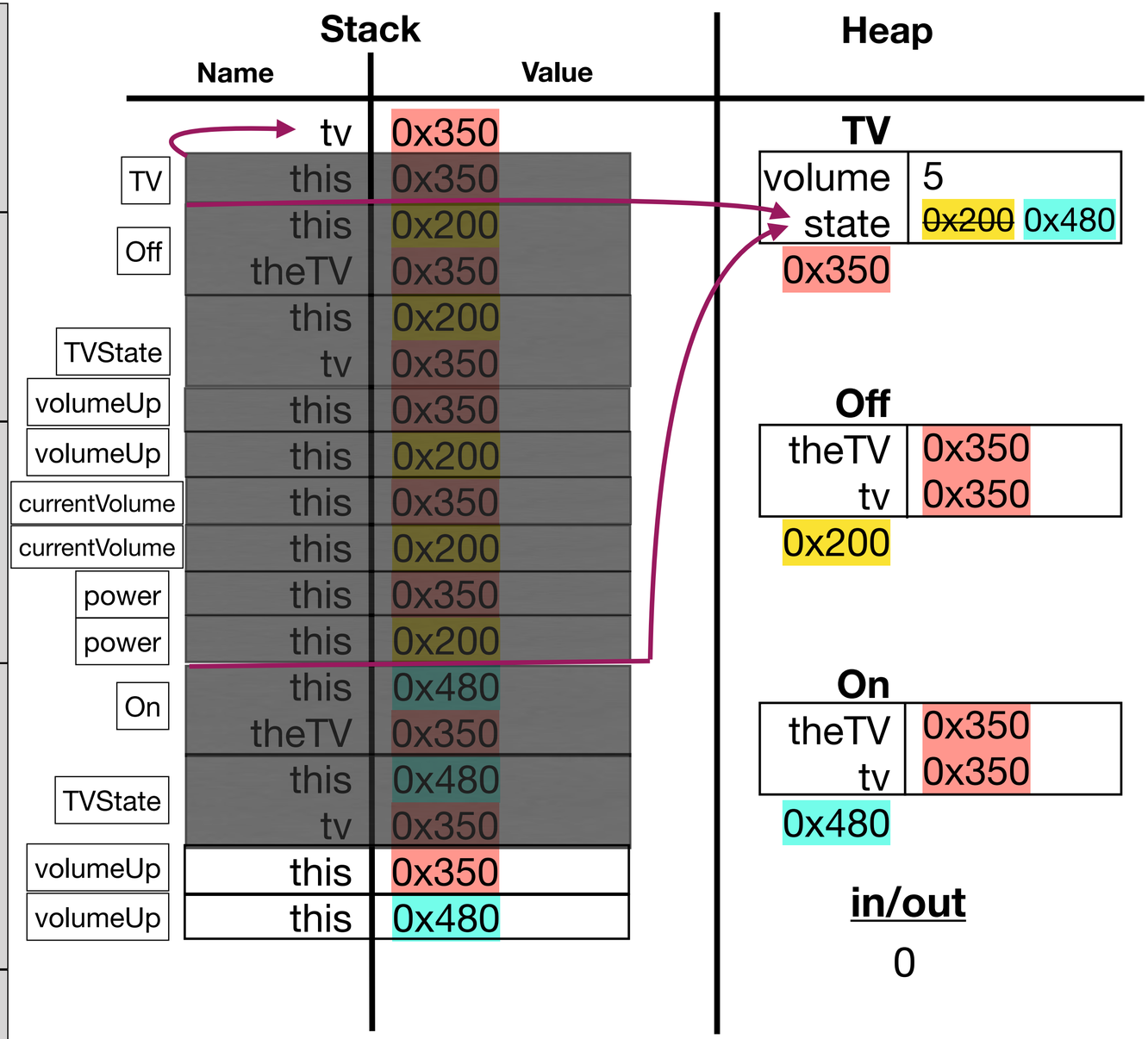
```scala
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```scala
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```scala
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```scala
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```scala
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Calling volumeUp again has different behavior
- The ***type*** of the state controls the behavior of the TV

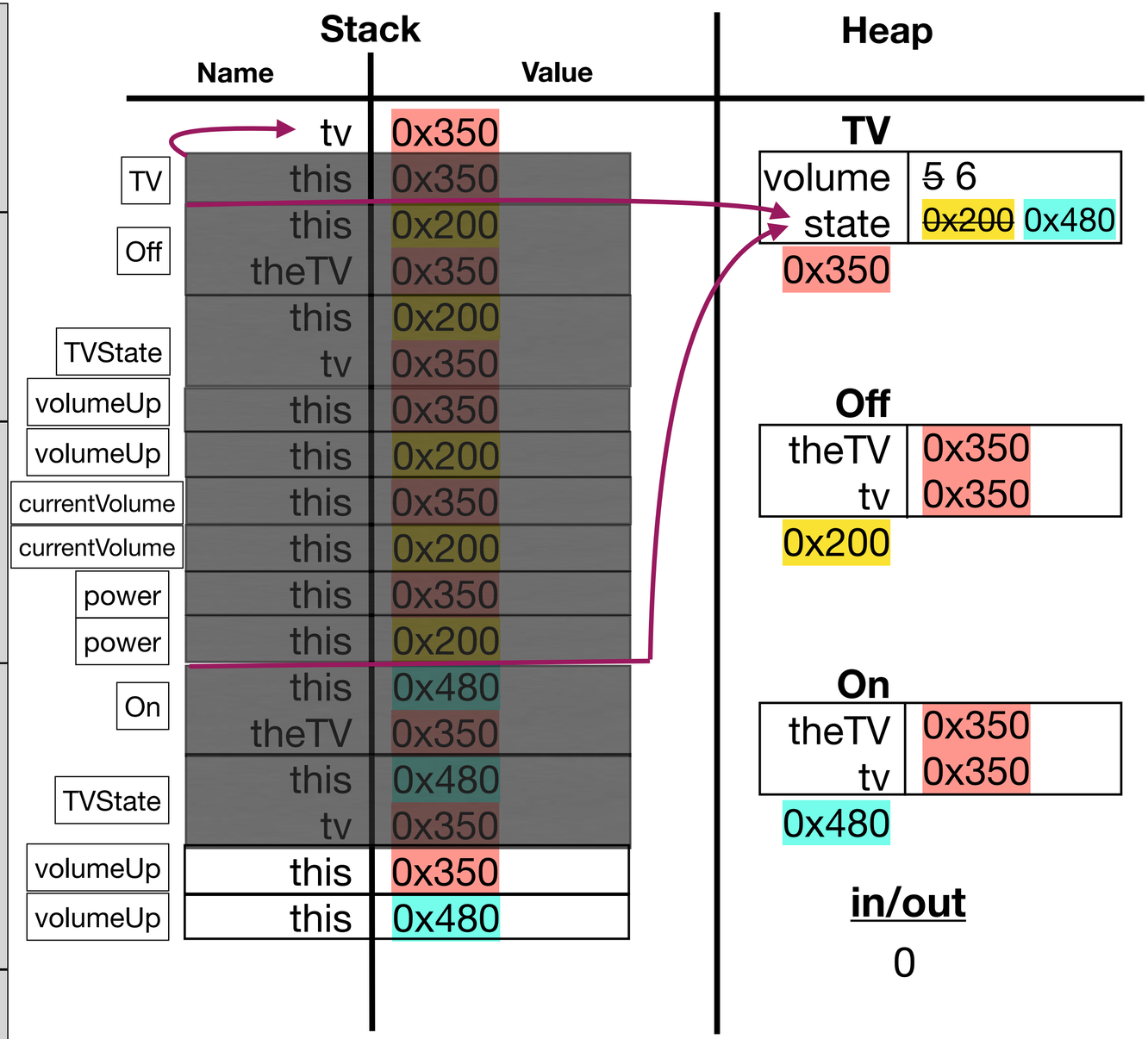
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Now that the TV is On, volumeUp increases the volume of the TV

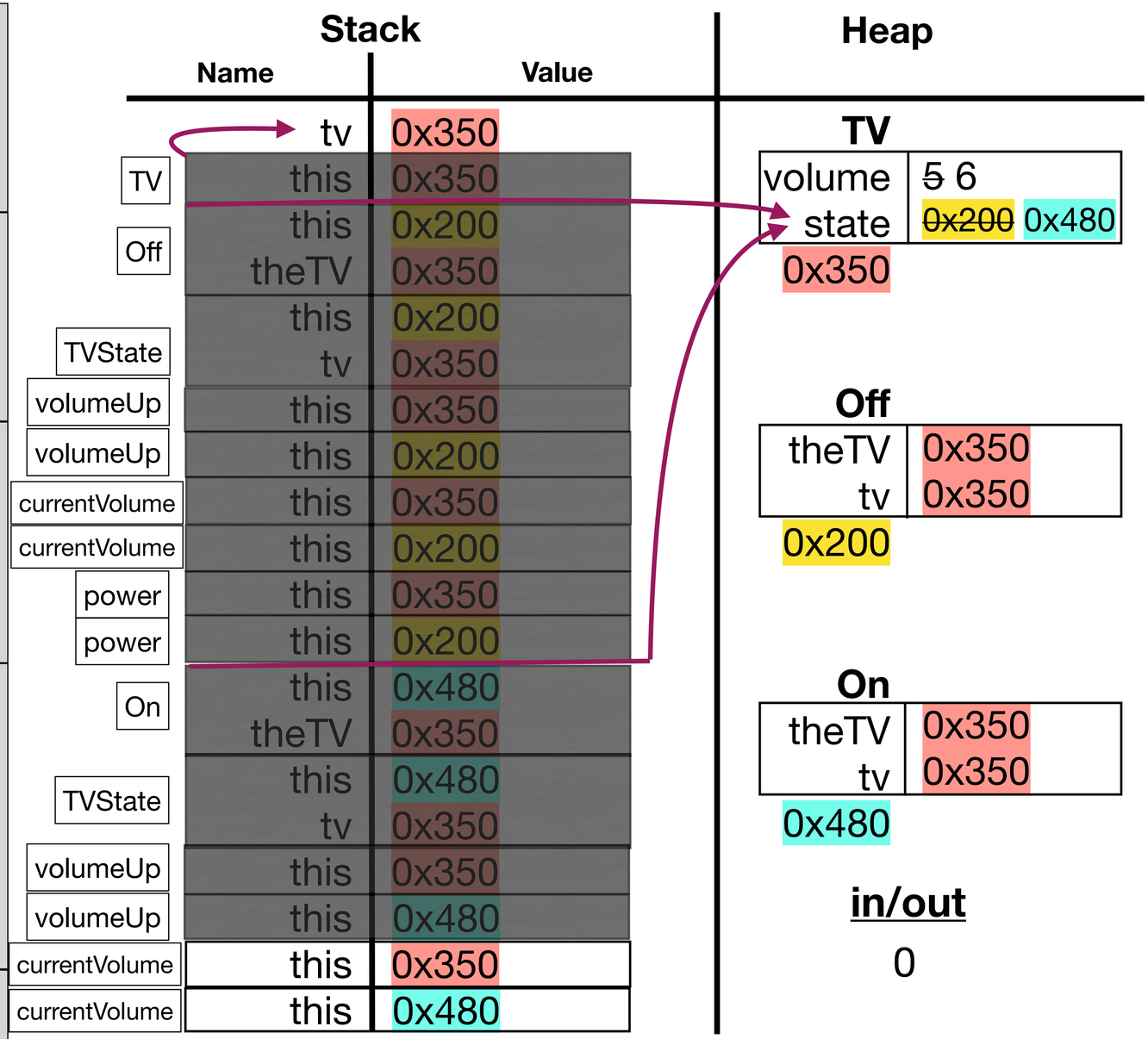
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- With the TV in the On state, currentVolume returns the volume of the TV

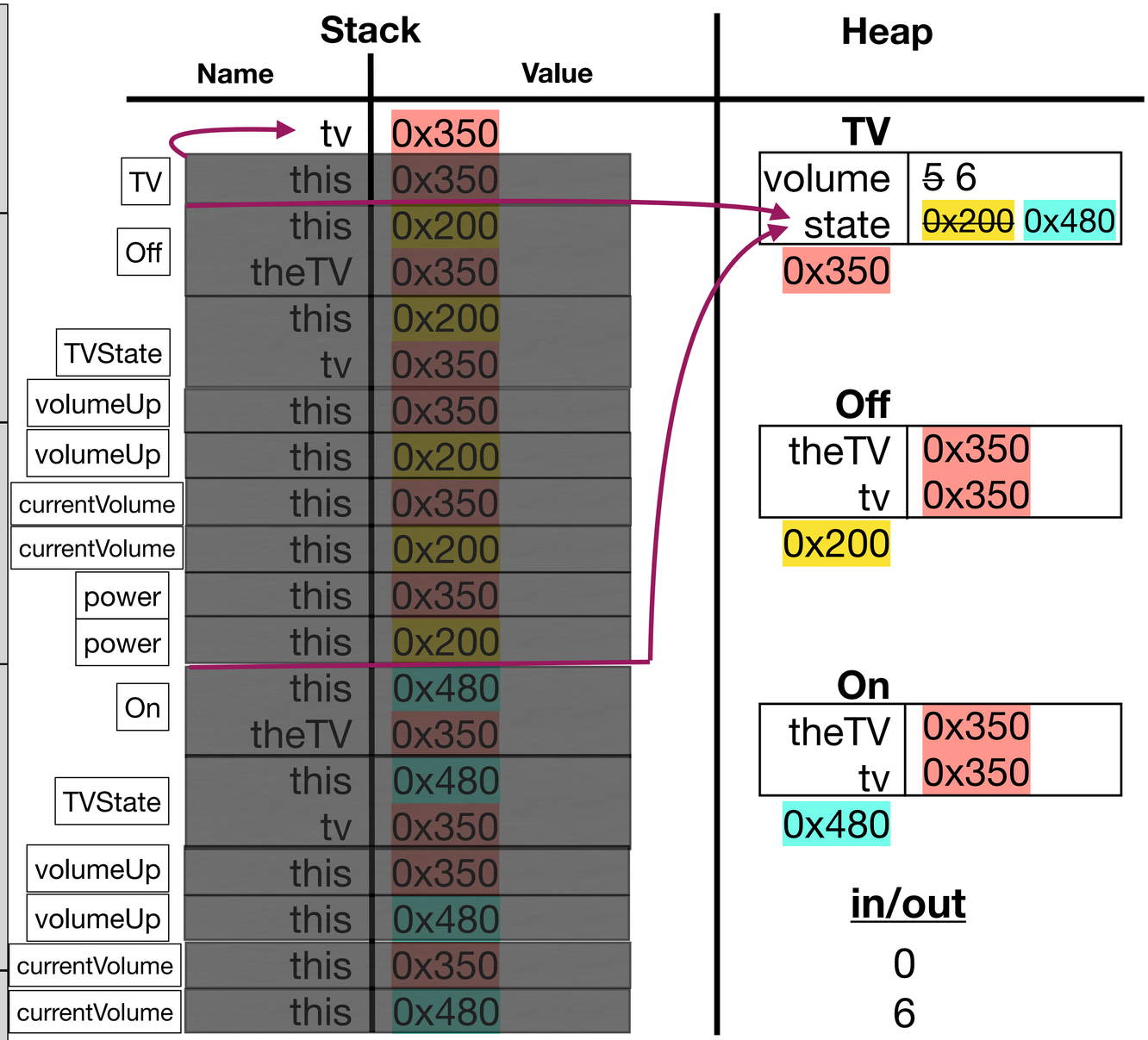
```
class On(theTV: TV) extends TVState(theTV) {
  override def volumeUp(): Unit = {this.tv.volume += 1}
  override def volumeDown(): Unit = {this.tv.volume -= 1}
  override def power(): Unit = {
    this.tv.state = new Off(this.tv)}
}
```

```
class Off(theTV: TV) extends TVState(theTV) {
  override def power(): Unit = {
    this.tv.state = new On(this.tv)
  }
  override def currentVolume(): Int = {0}
}
```

```
abstract class TVState(val tv: TV) {
  def volumeUp(): Unit = {}
  def volumeDown(): Unit = {}
  def mute(): Unit = {}
  def power(): Unit = {}
  def currentVolume(): Int = {this.tv.volume}
}
```

```
class TV {
  var volume = 5
  var state: TVState = new Off(this)
  def volumeUp(): Unit = {this.state.volumeUp()}
  def volumeDown(): Unit = {this.state.volumeDown()}
  def mute(): Unit ={this.state.mute()}
  def power(): Unit = {this.state.power()}
  def currentVolume(): Int ={this.state.currentVolume()}
}
```

```
def main(args: Array[String]): Unit = {
  val tv: TV = new TV()
  tv.volumeUp()
  println(tv.currentVolume())
  tv.power()
  tv.volumeUp()
  println(tv.currentVolume())
}
```



- Questions?

State Pattern - Closing Thoughts

State pattern trade-offs

- **Pros**
  - Organizes code when a single class can have very different behavior in different circumstances
  - Each implemented method is only concerned with the reaction to 1 event (API call) in 1 state
  - Easy to change or add new behavior after the state pattern is setup
- **Cons**
  - Can add complexity if there are only a few states or if behavior does not change significantly across states
  - Spreading the behavior for 1 class across many classes can look complex and require clicking through many files to understand all the behavior

State Pattern - Closing Thoughts

- Do not use the state pattern everywhere
  - Decide if a class is complex enough to benefit from this pattern before applying it

- The state pattern in this class
  - I have to force you to use it by removing conditionals (Not realistic)
  - Used to reinforce your understanding of **inheritance** and **polymorphism**
  - Used as an example of a design pattern that can help organize your code
- When you're not forced to use this pattern
  - Weight the pros and cons to decide when it is the best approach