State Design Pattern

State Design Pattern

- Behavioral Pattern
- Allows object to alter its behavior when internal state changes
- Uses Polymorphism to define different behaviors for different states

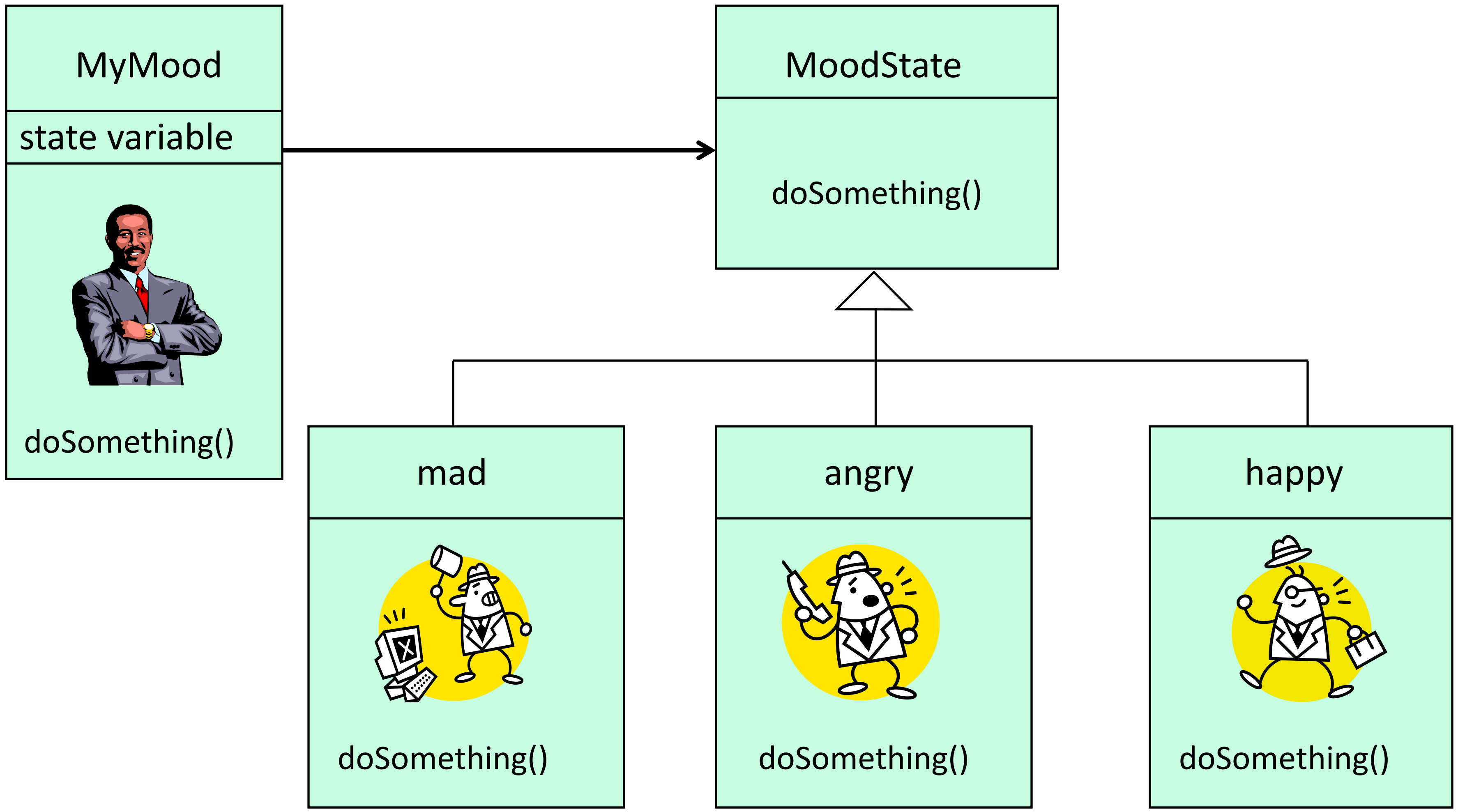
- Implementation of Replace conditional with Polymorphism

When to use STATE pattern ?

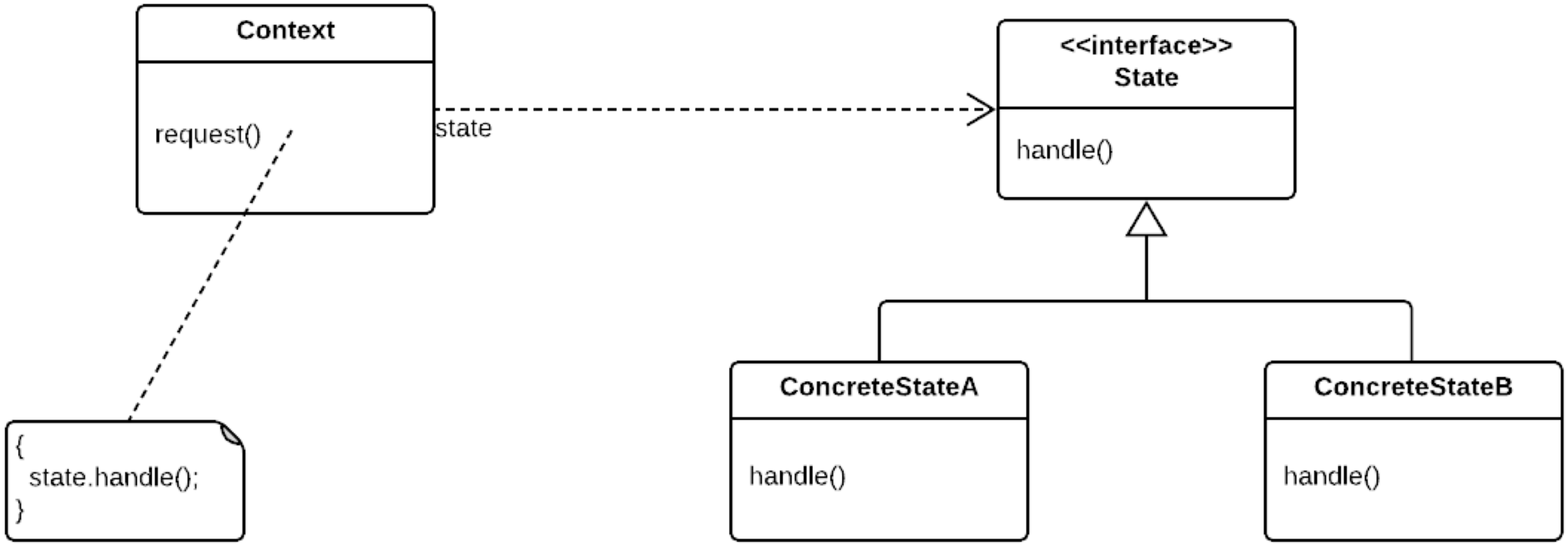
- State pattern is useful when there is an object that can be in one of several states, with different behavior in each state.
- To simplify operations that have large conditional statements that depend on the object’s state.

```
if (myself = happy) then
{
    eatIceCream();
    ….
    }
else if (myself = sad) then
{
    eatLotsOfIceCream();
    ….
  }
else if (myself = ecstatic) then
{
    ….
```

Example

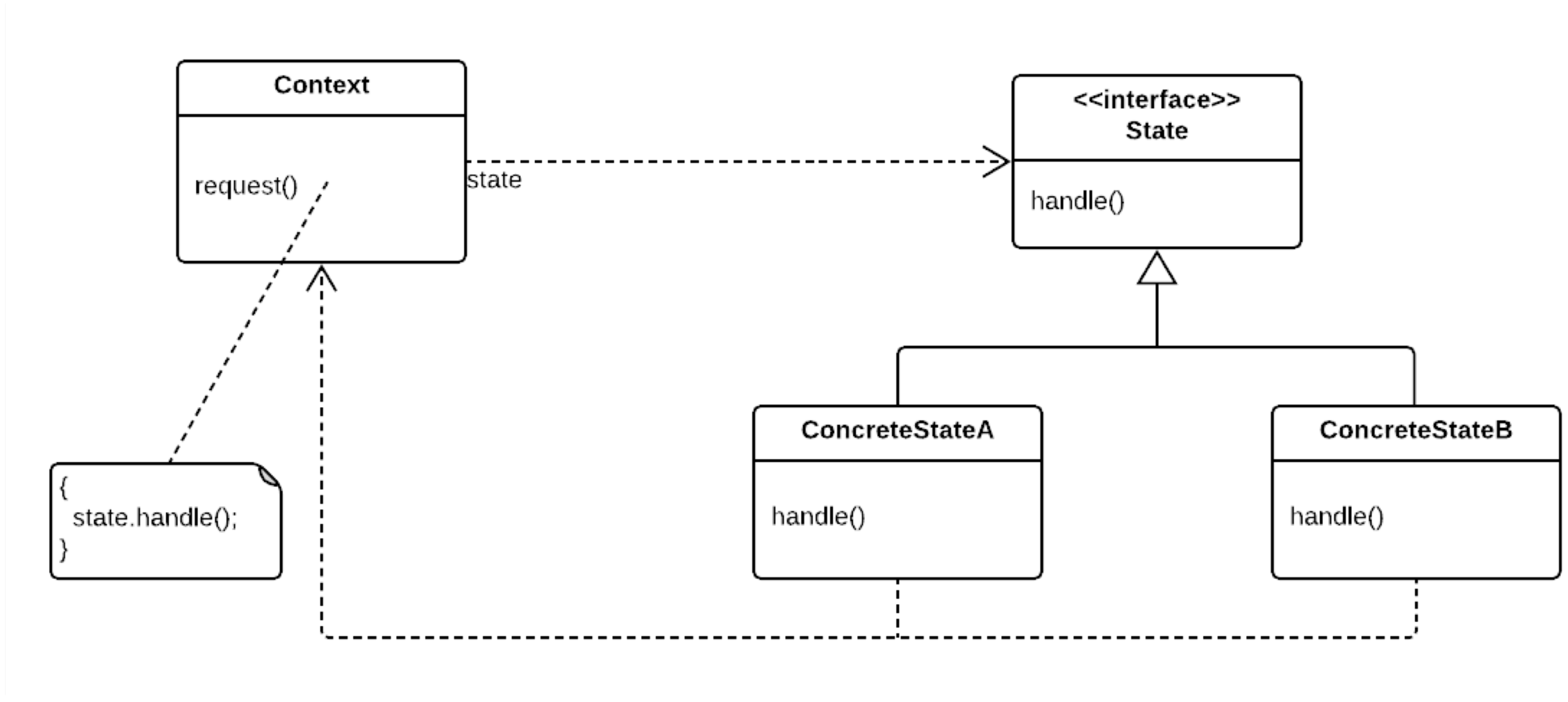

MyMood
state variable
doSomething()
MoodState
doSomething()
mad
doSomething()
angry
doSomething()
happy
doSomething()

State Pattern Structure



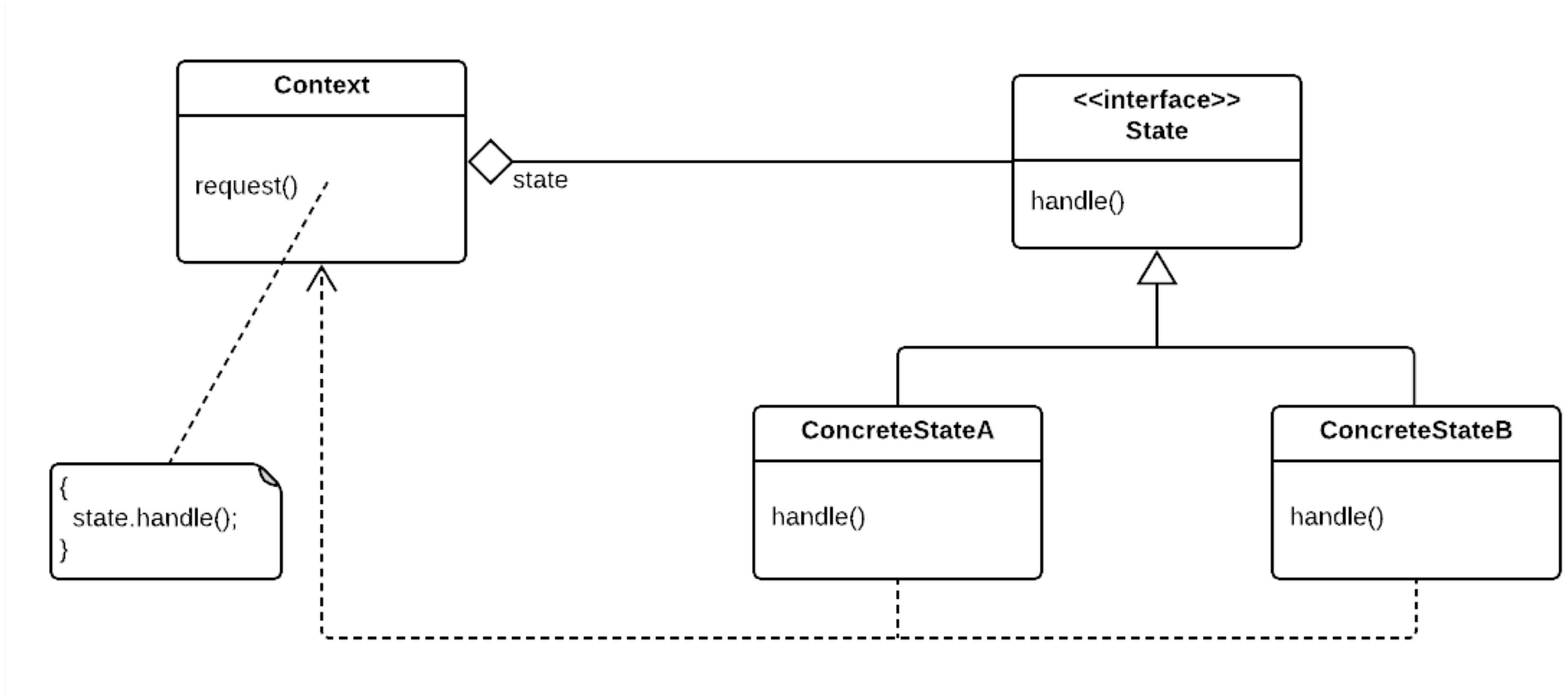
Allow an object to alter its behavior when its internal state changes.

State Pattern Structure



Allow an object to alter its behavior when its internal state changes.

State Pattern Structure



Allow an object to alter its behavior when its internal state changes.

```
// Not good: unwieldy "case-like" statement

class CeilingFanPullChain
{
   private int m_current_state;

   public CeilingFanPullChain()
   {
      m_current_state = 0;
   }
   public void pull()
   {
      if (m_current_state == 0)
      {
         m_current_state = 1;
         System.out.println("   low speed");
      }
      else if (m_current_state == 1)
      {
         m_current_state = 2;
         System.out.println("   medium speed");
      }
      else if (m_current_state == 2)
      {
         m_current_state = 3;
         System.out.println("   high speed");
      }
      else
      {
         m_current_state = 0;
         System.out.println("   turning off");
      }
   }
}
```

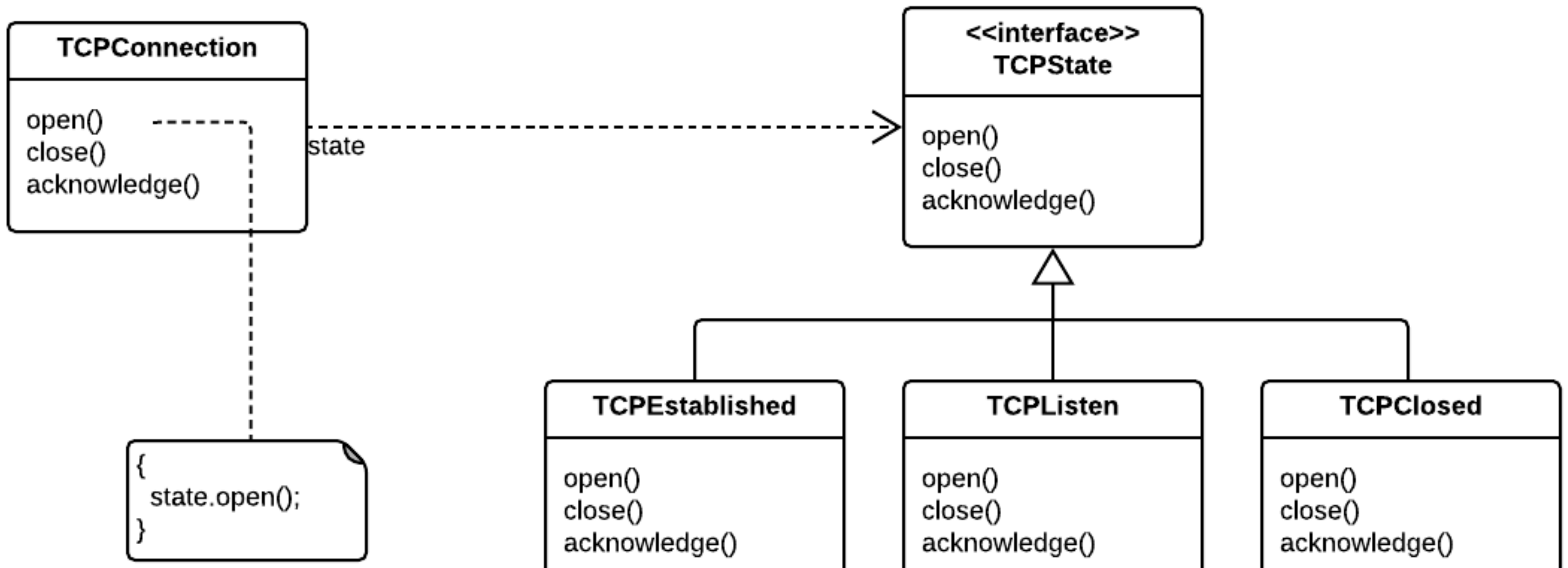
```
public class StateDemo
{
   public static void main(String[] args)
   {
      CeilingFanPullChain chain = new CeilingFanPullChain();
      while (true)
      {
         System.out.print("Press Enter");
         acceptInput();
         chain.pull();
      }
   }

   static void acceptInput()
   {
      BufferedReader in = new BufferedReader(
                            new InputStreamReader(System.in)) ;
      try
      {
         in.readLine();
      }
      catch (IOException ex)
      {
      }
   }
}
```

```java
class CeilingFanPullChain
{
   private State m_current_state;

   public CeilingFanPullChain()
   {
      m_current_state = new Off();
   }
   public void set_state(State s)
   {
      m_current_state = s;
   }
   public void pull()
   {
      m_current_state.pull(this);
   }
}
```

```java
interface State
{
  void pull(CeilingFanPullChain context);
}

class Off implements State
{
   public void pull(CeilingFanPullChain context)
   {
      context.set_state(new Low());
      System.out.println("   low speed");
   }
}

class Low implements State
{
   public void pull(CeilingFanPullChain context)
   {
      context.set_state(new Medium());
      System.out.println("   medium speed");
   }
}

class Medium implements State
{
   public void pull(CeilingFanPullChain context)
   {
      context.set_state(new High());
      System.out.println("   high speed");
   }
}

class High implements State
{
   public void pull(CeilingFanPullChain context)
   {
      context.set_state(new Off());
      System.out.println("   turning off");
   }
}
```

```
public class StateDemo
{
   public static void main(String[] args)
   {
      CeilingFanPullChain chain = new CeilingFanPullChain();
      while (true)
      {
         System.out.print("Press Enter");
         acceptInput();
         chain.pull();
      }
   }

   static void acceptInput()
   {
      BufferedReader in = new BufferedReader(new InputStreamReader(System.in) ;

      try
      {
         in.readLine();
      }
      catch (IOException ex)
      {
      }
   }
}
```

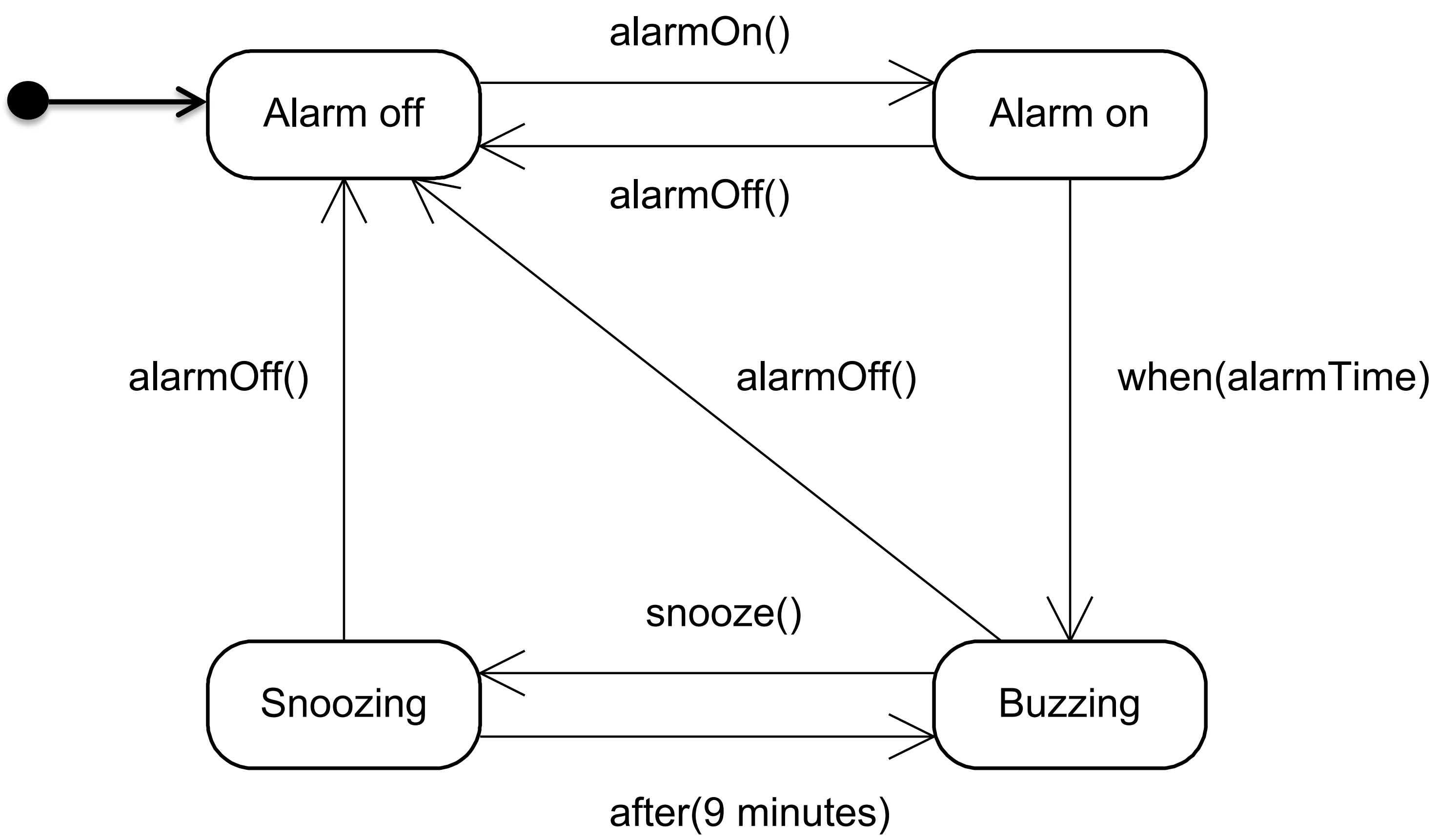
State Design Pattern



Alarm Clock State Machine

Clock State: *time*, *alarmTime*

Clock Inputs: *setTime()*, *setAlarmTime()*, *alarmOn()*, *alarmOff()*, *snooze()*


alarmOn()
Alarm off
Alarm on
alarmOff()
alarmOff()
alarmOff()
when(alarmTime)
snooze()
Snoozing
Buzzing
after(9 minutes)