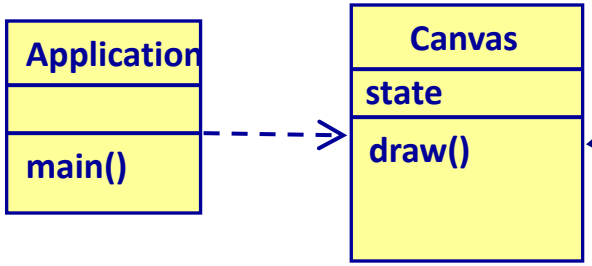
LESSON 5
STATE PATTERN

State pattern

- The state pattern is a design pattern that allows an object to completely change its behavior depending upon its current internal state.

Without state

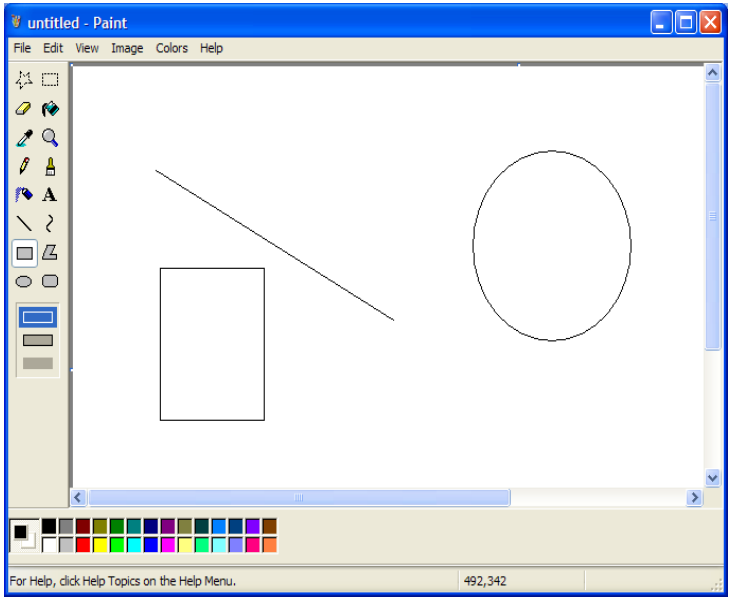



```
public void draw(){
      if (state.equals("line"))
        // draw a line
       else
        if (state.equals("rectangle"))
          // draw a rectangle
         else
          if (state.equals("circle"))
            // draw a circle
            else
              if (state.equals("triangle"))
                // draw a triangle
  }
```

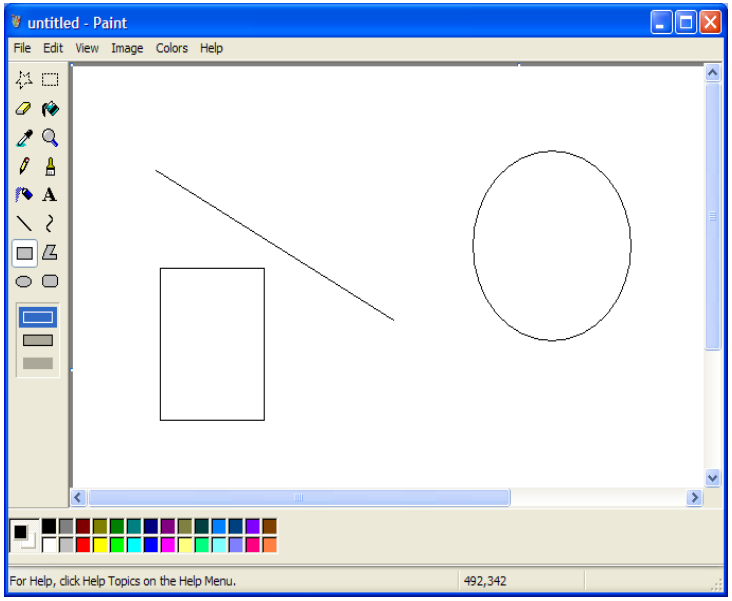
Complex if-then-else logic

Add a new state: you need to change the Canvas class

Logic of drawing shapes is not reusable for other classes

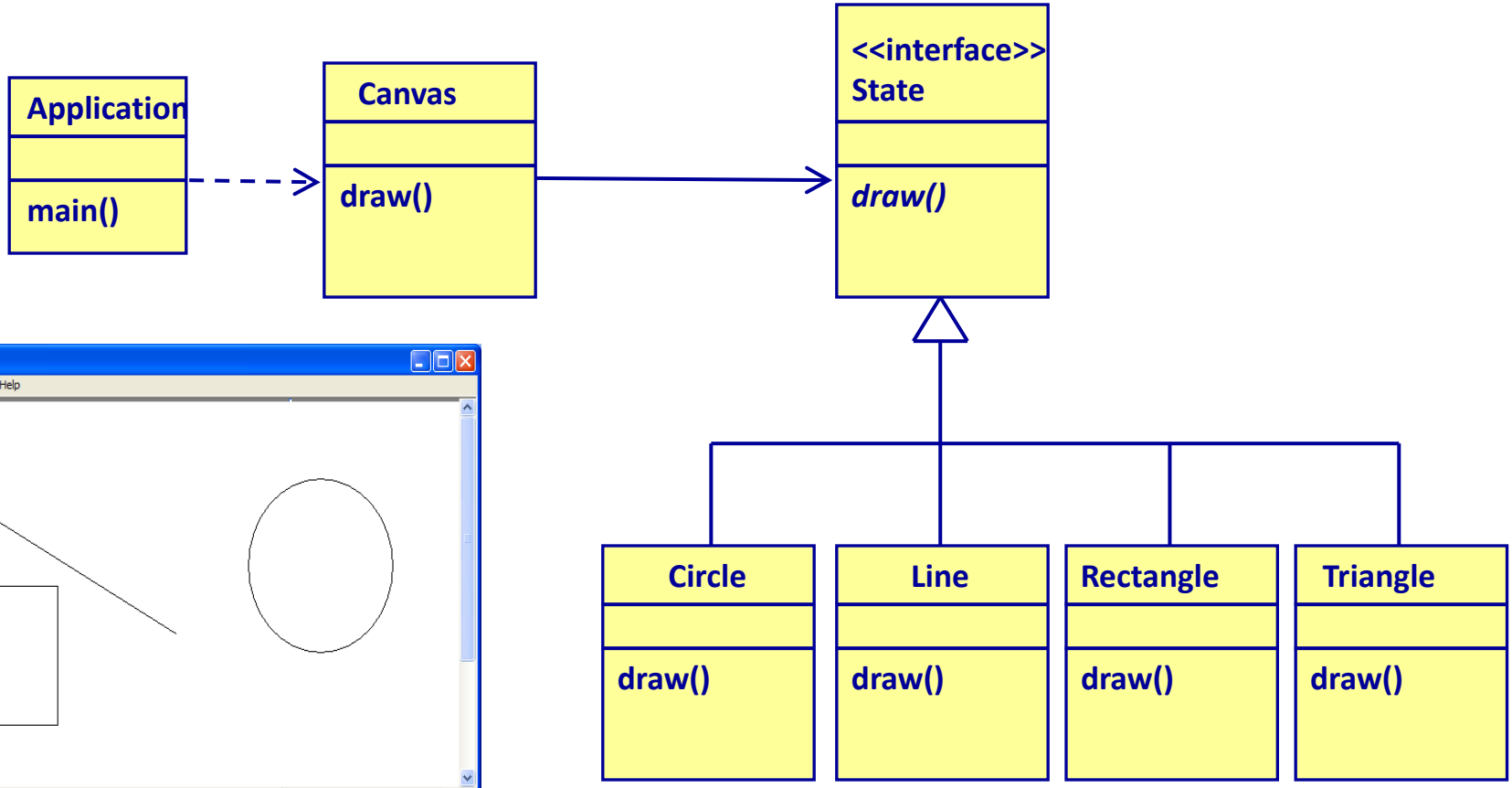


With state


untitled - Paint
File Edit View Image Colors Help
For Help, click Help Topics on the Help Menu.
492,342

No if-then-else logic
Application
main()
Canvas
draw()
<<interface>>
State
draw()
Specific state logic is encapsulated
Circle
draw()
Line
draw()
Rectangle
draw()
Triangle
draw()
Logic of drawing shapes is reusable for other classes
Easy to add a new state without changing the canvas class

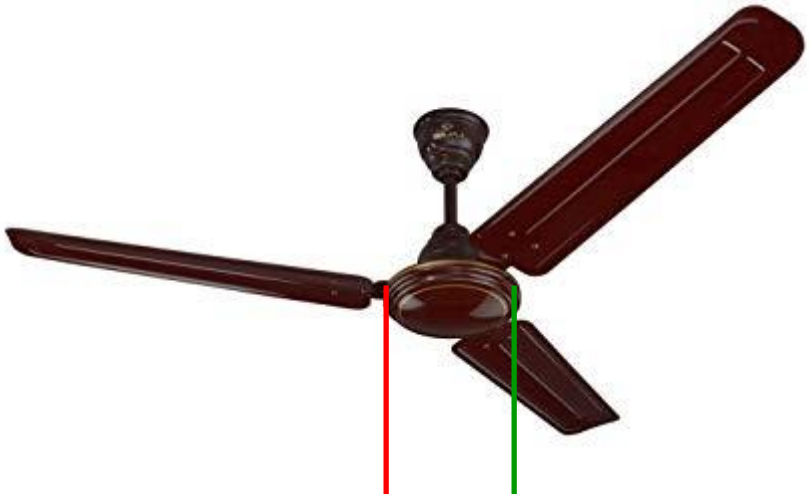
State or strategy?

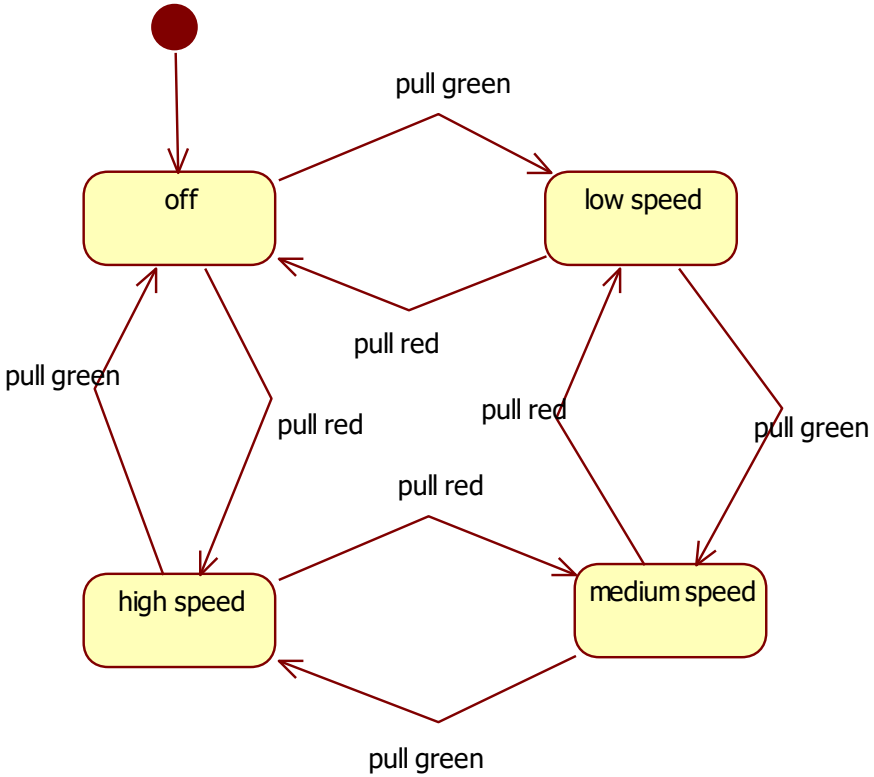


State or strategy?

- Strategy
  - The context can have different algorithms
  - Strategies do not know each other
    - The context has one strategy
- State
  - The context can have different states
  - States know the local state transitions
    - The context has currently one state, but that state will change over time.

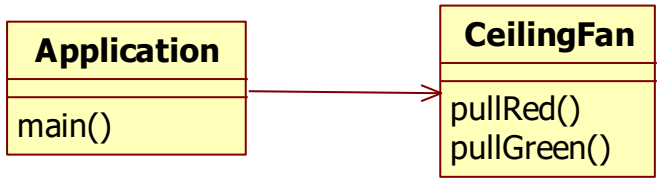
Ceiling fan




pull green
off
low speed
pull red
pull green
pull red
pull red
pull green
pull red
high speed
medium speed
pull green

Without state

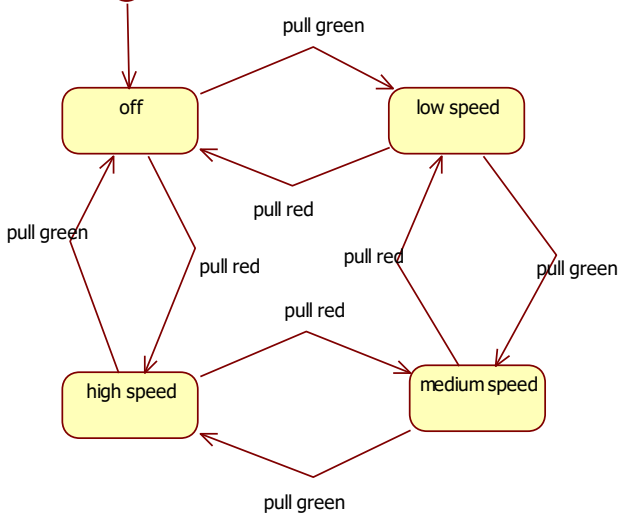
```
public class Application {
  public static void main(String[] args) {
    CeilingFan fan = new CeilingFan();
    fan.pullgreen();
    fan.pullgreen();
    fan.pullred();
    fan.pullred();
  }
}
```

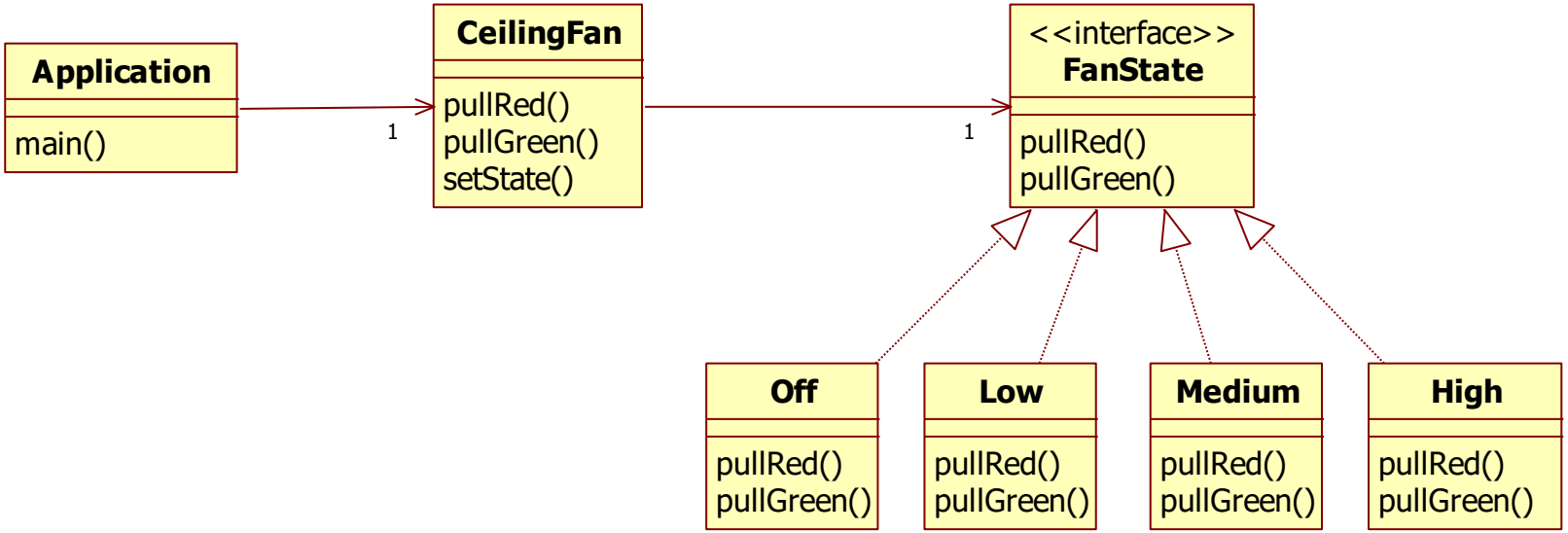



```
public class CeilingFan {
  int current_state = 0;

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

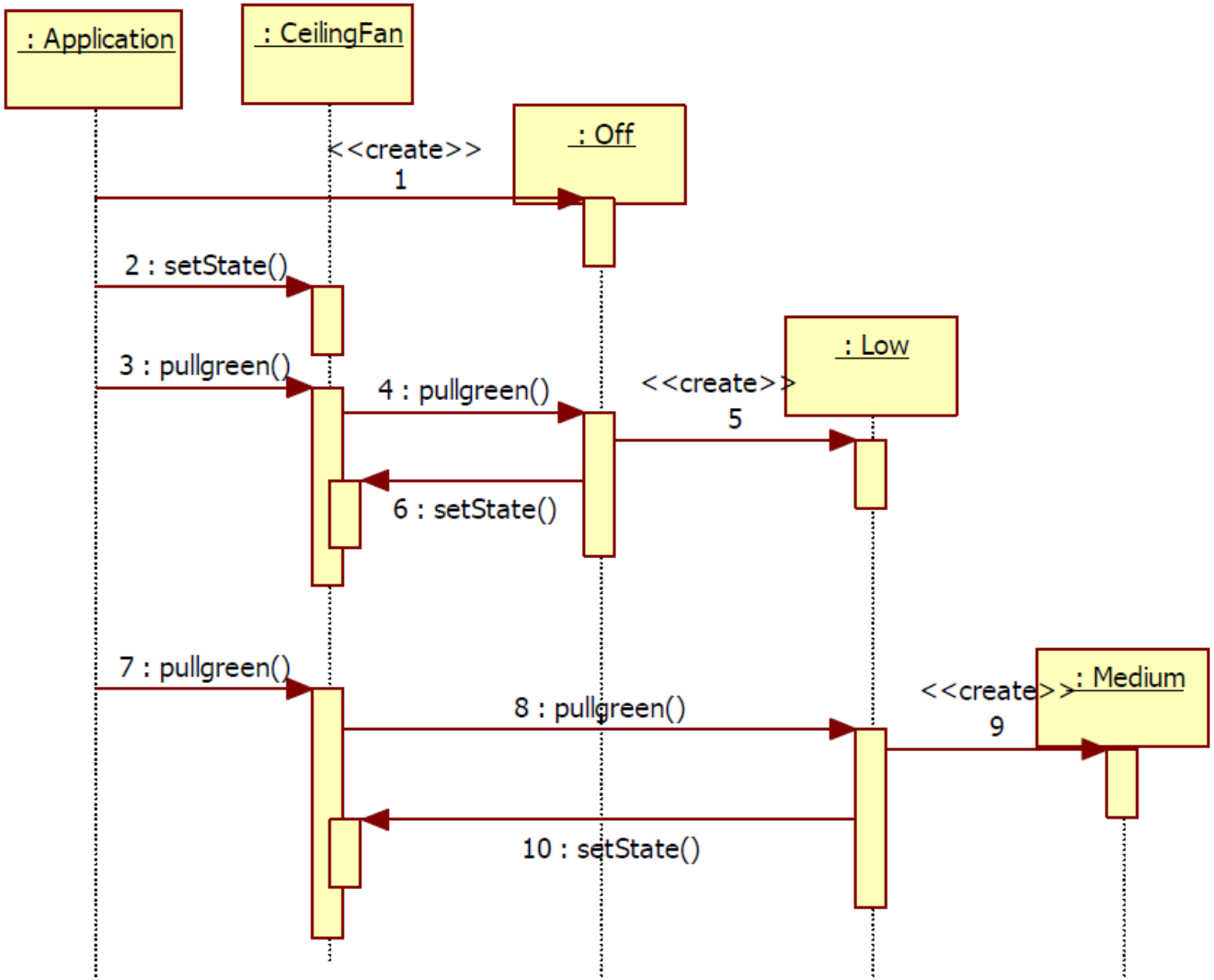
```
  public void pullred() {
    if (current_state == 0) {
      current_state = 3;
      System.out.println("high speed");
    } else if (current_state == 1) {
       current_state = 0;
       System.out.println("turning off");
    } else if (current_state == 2) {
      current_state = 1;
      System.out.println("low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

With state





With state



With state

```
public class Application {
  public static void main(String[] args) {
    CeilingFan fan = new CeilingFan();
    fan.setState(new Off(fan, true));
    fan.pullgreen();
    fan.pullgreen();
    fan.pullred();
    fan.pullred();
  }
}
```

```
public class CeilingFan {
  FanState state;

  public void setState(FanState state) {
    this.state = state;
  }

  public void pullgreen() {
    state.pullgreen();
  }

  public void pullred() {
    state.pullred();
  }
}
```

```
public interface FanState {
   void pullred();
   void pullgreen();
}
```

With state

```
public class Off implements FanState{
  CeilingFan fan;

  public Off(CeilingFan fan, boolean start) {
    this.fan=fan;
    if (!start)
     System.out.println( "turning off" );
  }

  public void pullgreen() {
    Low newstate = new Low(fan);
    fan.setState(newstate);
  }

  public void pullred() {
    High newstate = new High(fan);
    fan.setState(newstate);
  }
}
```

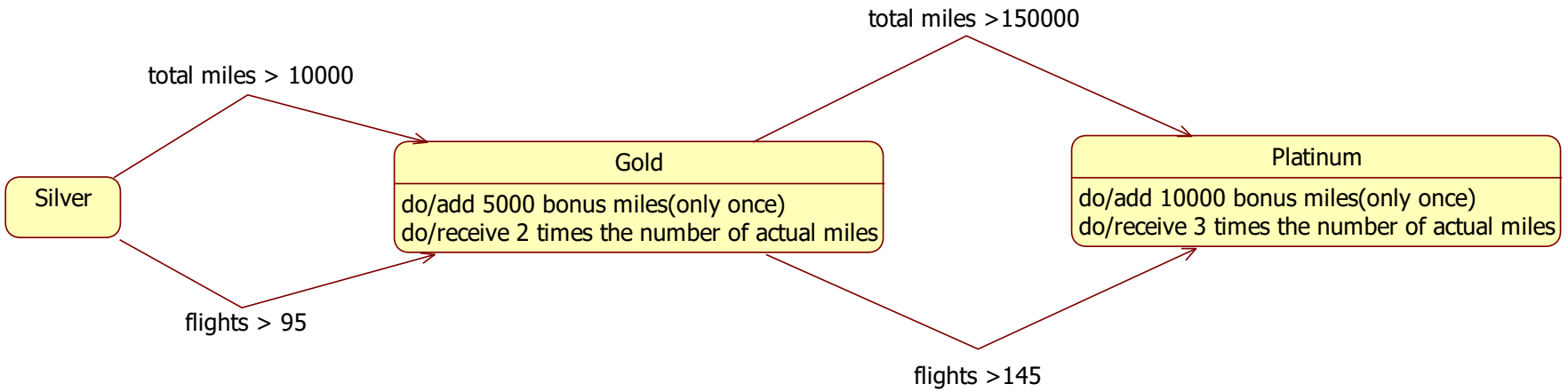
```
public class Low implements FanState{
  CeilingFan fan;

  public Low(CeilingFan fan) {
    this.fan=fan;
    System.out.println( "Low speed" );
  }

  public void pullgreen() {
    Medium newstate = new Medium(fan);
    fan.setState(newstate);
  }

  public void pullred() {
    Off newstate = new Off(fan, false);
    fan.setState(newstate);
  }
}
```

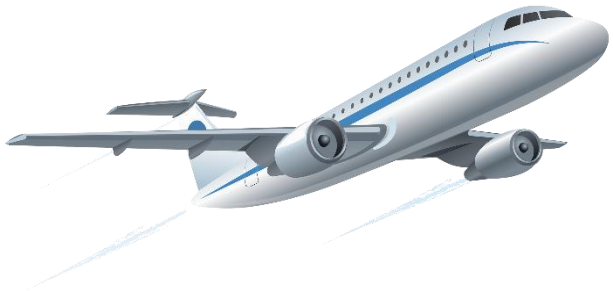
With state

```
public class Medium implements FanState{
  CeilingFan fan;

  public Medium(CeilingFan fan) {
    this.fan=fan;
    System.out.println( "medium speed" );
  }

  public void pullgreen() {
    High newstate = new High(fan);
    fan.setState(newstate);
  }

  public void pullred() {
    Low newstate = new Low(fan);
    fan.setState(newstate);
  }
}
```

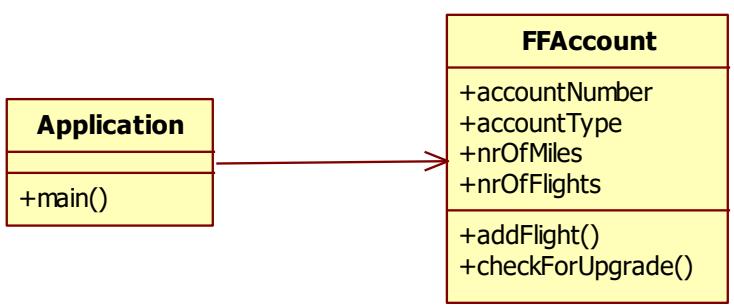
```
public class High implements FanState{
  CeilingFan fan;

  public High(CeilingFan fan) {
    this.fan=fan;
    System.out.println( "high speed" );
  }

  public void pullgreen() {
    Off newstate = new Off(fan, false);
    fan.setState(newstate);
  }

  public void pullred() {
      Medium newstate = new Medium(fan);
      fan.setState(newstate);
  }
}
```

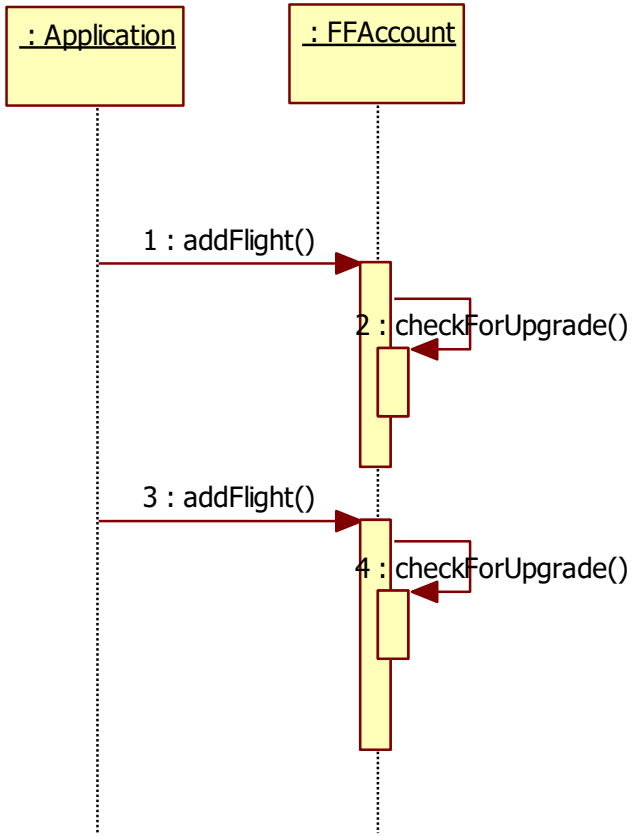
Frequent flyer account





Without state





Without state

```
public class FFAccount {
  private String accountNumber;
  private String accountType;
  private int numberOfMiles;
  private int numberOfFlights;

  public FFAccount(String aNumber, String accountType) {
    this.accountNumber = aNumber;
    this.accountType = accountType;
  }

  public void addFlight(int newMiles) {
    if (accountType.equals("silver")) {
      numberOfMiles += newMiles;
      numberOfFlights++;
      checkForUpgrade();
    } else {
      if (accountType.equals("gold")) {
        numberOfMiles += (2 * newMiles);
        numberOfFlights++;
        checkForUpgrade();
      } else {
        if (accountType.equals("platinum")) {
          numberOfMiles += (3 * newMiles);
          numberOfFlights++;
        }
      }
    }
  }
```

Without state

```
public void checkForUpgrade() {
  if (accountType.equals("silver") && (numberOfMiles > 100000) || (numberOfFlights > 95)) {
    accountType = "gold";
    numberOfMiles += 5000;
  }
  if (accountType.equals("gold") && (numberOfMiles > 150000) || (numberOfFlights > 145)) {
    accountType = "platinum";
    numberOfMiles += 10000;
  }
}
```

Without state

```
public class Application {

  public static void main(String[] args) {
    FFAccount ffaccount =  new FFAccount("213425", "silver");
    ffaccount.addFlight(13000);
    System.out.println("Accountnr ="+ffaccount.getAccountNumber());
    System.out.println("Account type ="+ffaccount.getAccountType());
    System.out.println("miles ="+ffaccount.getNumberOfMiles());

    ffaccount.addFlight(99000);
    System.out.println("Accountnr ="+ffaccount.getAccountNumber());
    System.out.println("Account type ="+ffaccount.getAccountType());
    System.out.println("miles ="+ffaccount.getNumberOfMiles());
  }
}
```

```
Accountnr =213425
Account type =silver
miles =13000
Accountnr =213425
Account type =gold
miles =112000
```

Problem

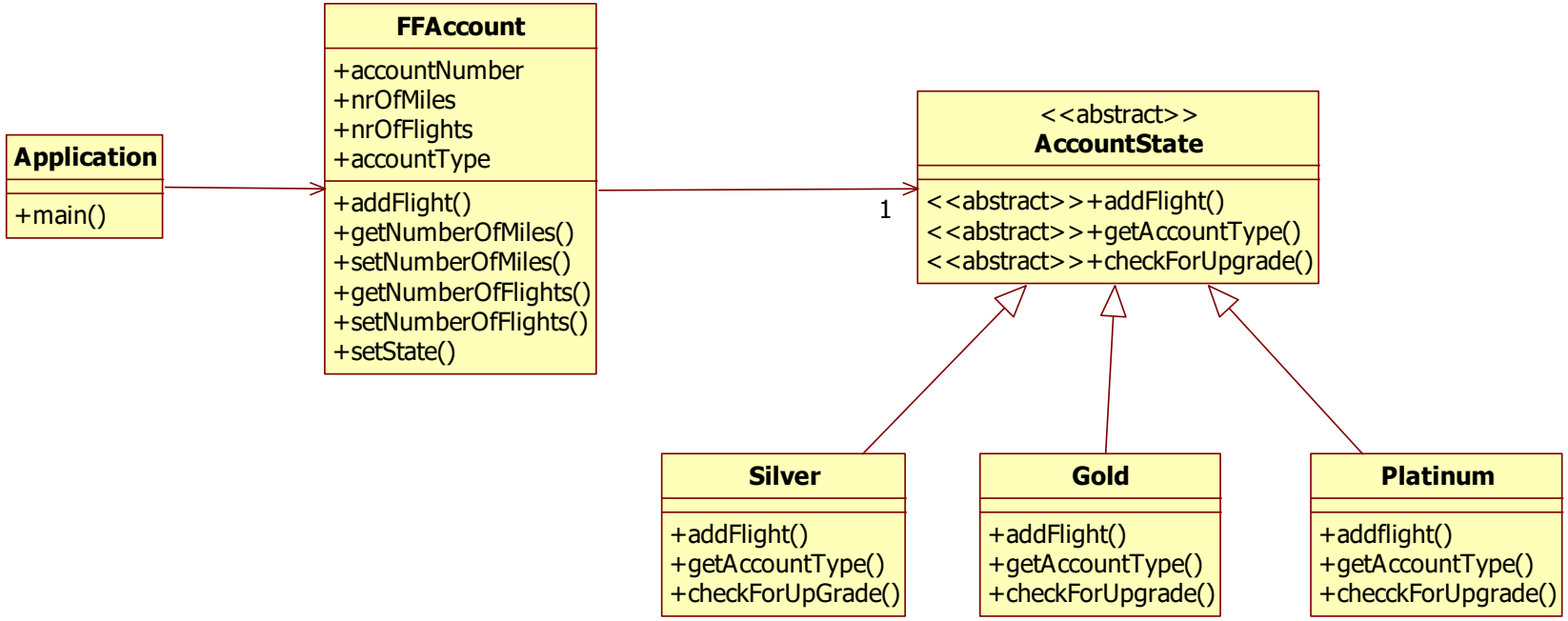
```
public class FFAccount {
...
public void addFlight(int newMiles) {
  if (accountType.equals("silver")) {
    numberOfMiles += newMiles;
    numberOfFlights++;
    checkForUpgrade();
  } else {
    if (accountType.equals("gold")) {
      numberOfMiles += (2 * newMiles);
      numberOfFlights++;
      checkForUpgrade();
    } else {
      if (accountType.equals("platinum")) {
        numberOfMiles += (3 * newMiles);
        numberOfFlights++;
      }
    }
  }
}
public void checkForUpgrade() {
  if (accountType.equals("silver") && (numberOfMiles > 100000) || (numberOfFlights > 95)) {
    accountType = "gold";
    numberOfMiles += 5000;
  }
  if (accountType.equals("gold") && (numberOfMiles > 150000) || (numberOfFlights > 145)) {
    accountType = "platinum";
    numberOfMiles += 10000;
  }
}
```

Add a new state: you need to change the FFAccount class

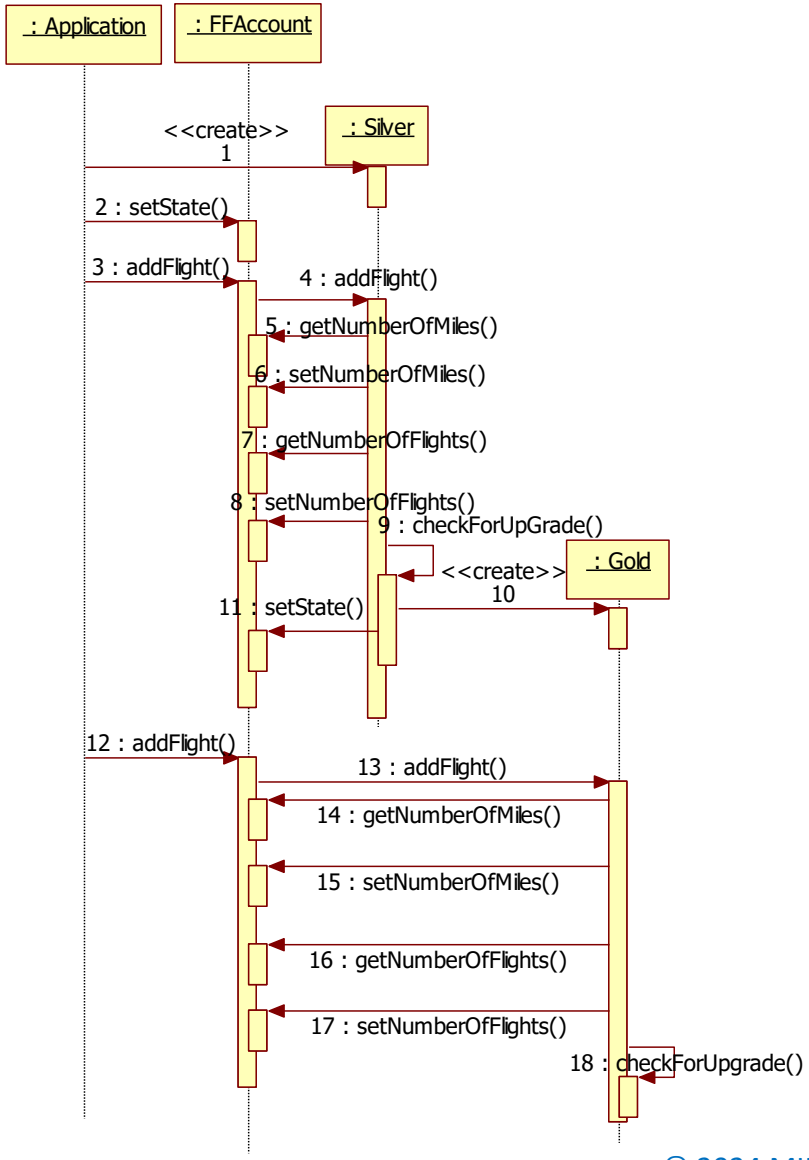
Complex if-then-else logic

Logic of state is not reusable for other classes

With state



With state



With state

```
public class FFAccount {
  private String accountNumber;
  private int numberOfMiles;
  private int numberOfFlights;
  private AccountState accountState;

  public FFAccount(String aNumber) {
    accountNumber=aNumber;
  }

  public void addFlight(int newMiles){
    accountState.addFlight(newMiles);
  }
  public String getAccountType() {
    return accountState.getAccountType();
  }

 ...
}
```

```
public abstract class AccountState {
  protected final FFAccount account;
  public AccountState(FFAccount account) {
    this.account=account;
  }
  public abstract void addFlight(int newMiles);
  public abstract String getAccountType();
}
```

Silver state

```
public class Silver extends AccountState{
  public Silver(FFAccount account) {
    super(account);
  }
  public void addFlight(int newMiles){
    account.setNumberOfMiles(account.getNumberOfMiles()+newMiles);
    account.setNumberOfFlights(account.getNumberOfFlights()+1);
    checkForUpgrade();
  }

  public void checkForUpgrade(){
    if ((account.getNumberOfMiles() > 100000)||
                    (account.getNumberOfFlights() > 95)){
      AccountState newState = new Gold(account) ;
      account.setAccountState(newState);
    }
  }

  public String getAccountType() {
    return "Silver";
  }
}
```

Gold state

```
public class Gold extends AccountState {
  public Gold(FFAccount account) {
    super(account);
  }

  public void addFlight(int newMiles){
    account.setNumberOfMiles(account.getNumberOfMiles()+(2*newMiles));
    account.setNumberOfFlights(account.getNumberOfFlights()+1);
    checkForUpgrade();
  }

  public void checkForUpgrade(){
    if ((account.getNumberOfMiles() > 150000)||
                (account.getNumberOfFlights() > 145)){
      AccountState newState = new Platinum(account) ;
      account.setAccountState(newState);
    }
  }

  public String getAccountType() {
    return "Gold";
  }
}
```

Platinum state

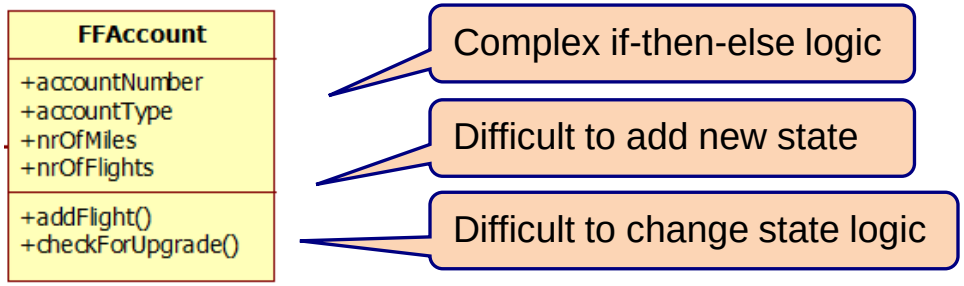
```
public class Platinum extends AccountState {
  public Platinum(FFAccount account) {
    super(account);
  }

  public void addFlight(int newMiles) {
    account.setNumberOfMiles(account.getNumberOfMiles() + (3 * newMiles));
    account.setNumberOfFlights(account.getNumberOfFlights() * 2);
  }

  public String getAccountType() {
    return "Platinum";
  }
}
```

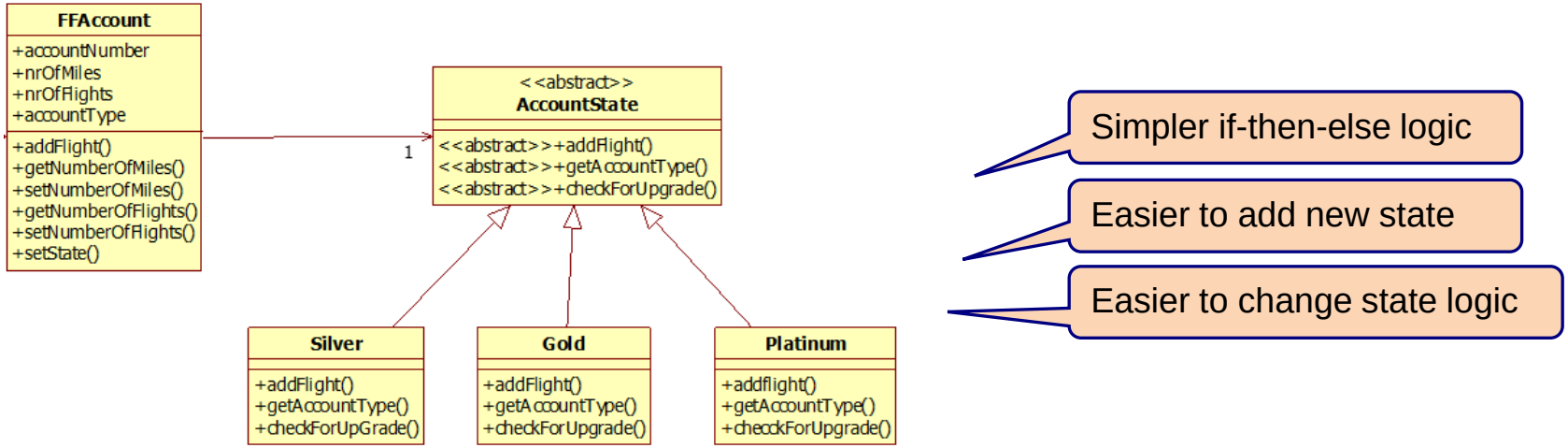
With state

```
public class Application {

  public static void main(String[] args) {
    FFAccount ffaccount =  new FFAccount("213425");
    AccountState accountState = new Silver(ffaccount);
    ffaccount.setAccountState(accountState);
    ffaccount.addFlight(13000);
    System.out.println("Accountnr ="+ffaccount.getAccountNumber());
    System.out.println("Account type ="+ffaccount.getAccountType());
    System.out.println("miles ="+ffaccount.getNumberOfMiles());

    ffaccount.addFlight(99000);
    System.out.println("Accountnr ="+ffaccount.getAccountNumber());
    System.out.println("Account type ="+ffaccount.getAccountType());
    System.out.println("miles ="+ffaccount.getNumberOfMiles());
  }
}
```

```
Accountnr =213425
Account type =Silver
miles =13000
Accountnr =213425
Account type =Gold
miles =112000
```

State advantages



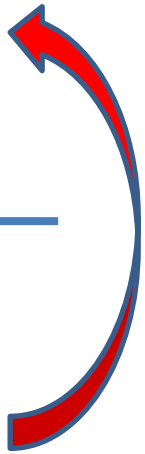


Main point

- The State Pattern allows an object to alter its behavior when its internal state changes
- Knowledge is different in different states of consciousness.

Connecting the parts of knowledge with the wholeness of knowledge

1. The state pattern can be applied whenever we have complex state logic.
2. The state pattern transforms complex if-then-else logic into many simpler if-then-else structures.



3. **Transcendental consciousness** is the source off all relative states.
4. **Wholeness moving within itself:** In Unity Consciousness, one experiences the unity between yourself and all of creation.