CS 525 - ASD

Advanced Software Development

MS.CS Program

Department of Computer Science

Rene de Jong, MsC.

Maharishi University
OF MANAGEMENT

CS 525 - ASD

Advanced Software Development

Maharishi University
OF MANAGEMENT

Lesson 6

L1: ASD Introduction
L2: Strategy, Template method
L3: Observer pattern
L4: Composite pattern, iterator pattern
L5: Command pattern
L6: State pattern
L7: Chain Of Responsibility pattern

Midterm

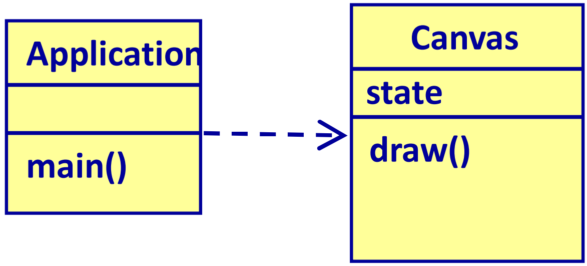
L8: Proxy, Adapter, Mediator
L9: Factory, Builder, Decorator, Singleton
L10: Framework design
L11: Framework implementation
L12: Framework example: Spring framework
L13: Framework example: Spring framework

Final

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

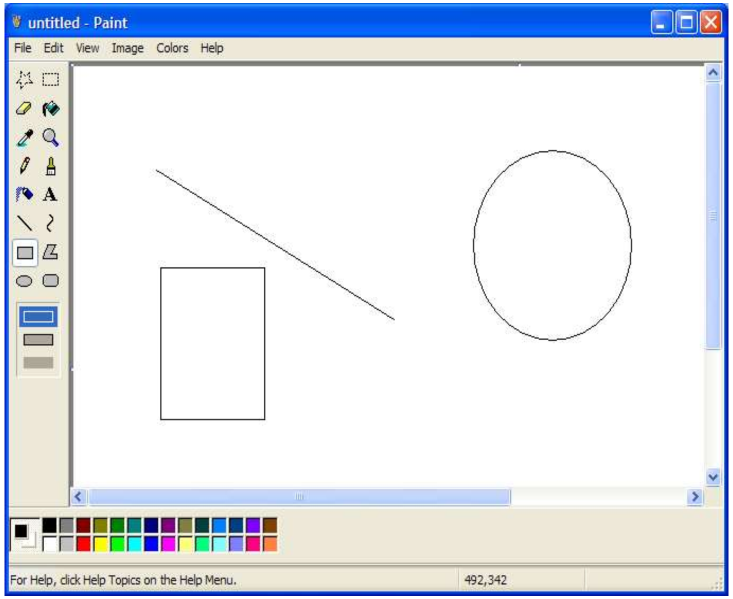


Complex if-then-else logic

```
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
   }
```

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

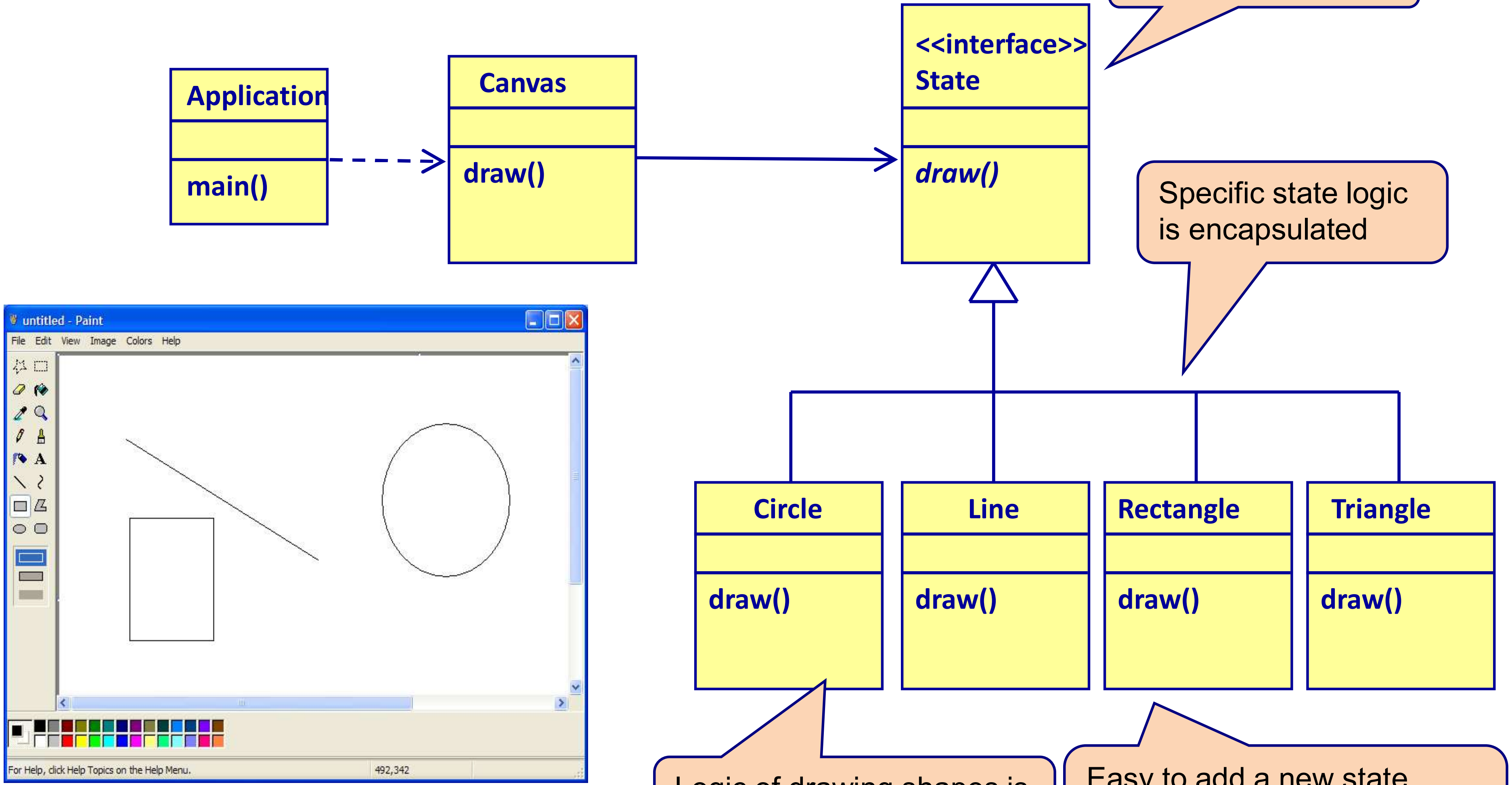
Logic of drawing shapes is not reusable for other classes



For Help, click Help Topics on the Help Menu.

492,342

With state



For Help, click Help Topics on the Help Menu.

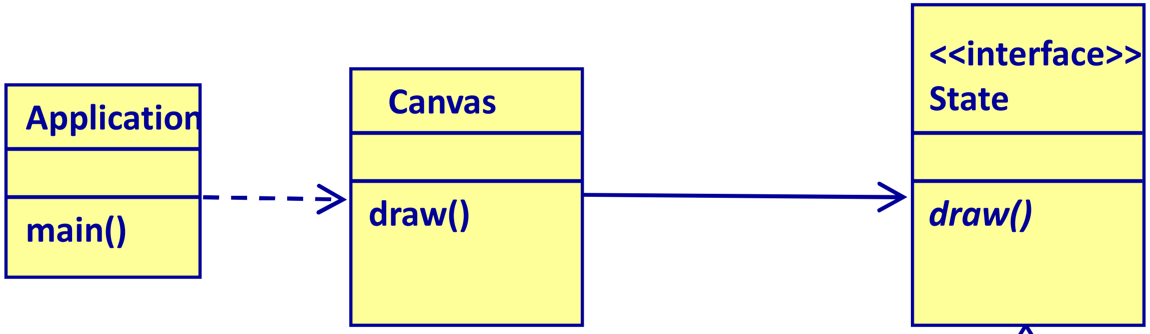
No if-then-else logic

Specific state logic is encapsulated

Logic of drawing shapes is reusable for other classes

Easy to add a new state without changing the canvas class

State or strategy?



Application

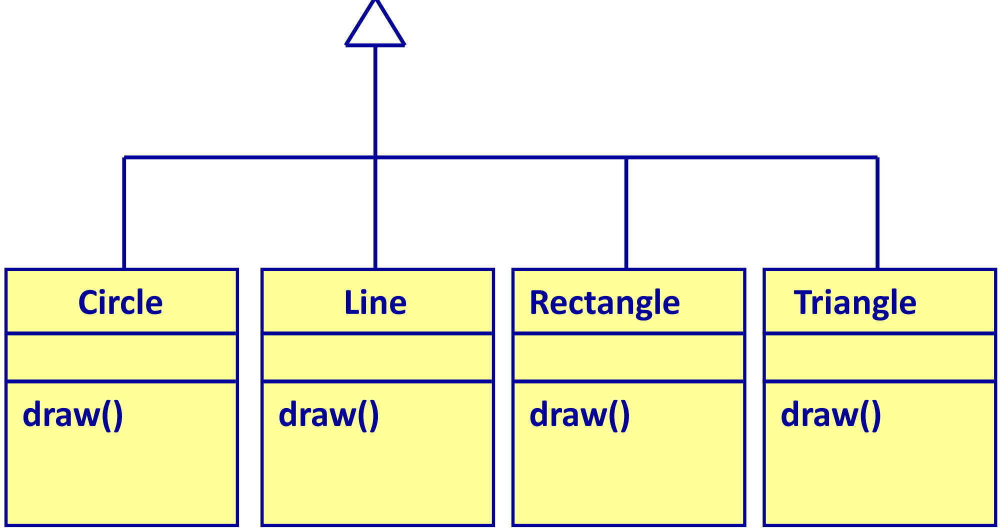
main()

Canvas

draw()

<<interface>>
State

draw()



Circle

draw()

Line

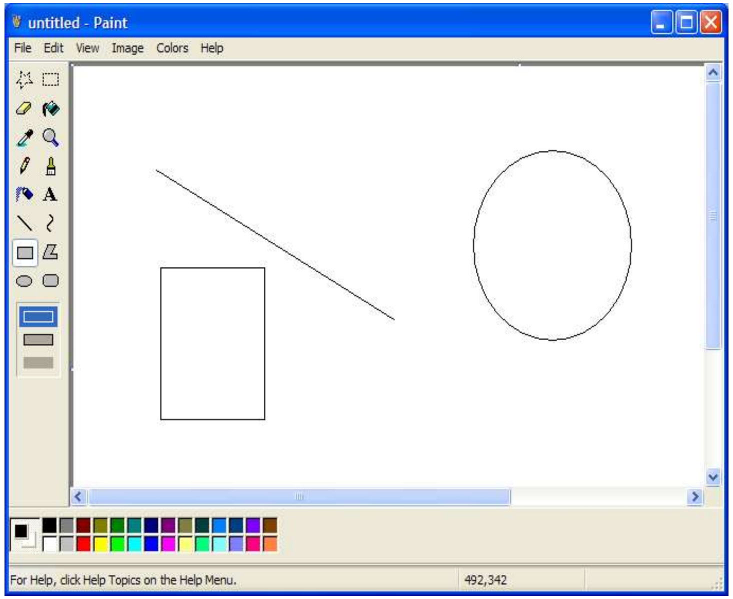
draw()

Rectangle

draw()

Triangle

draw()



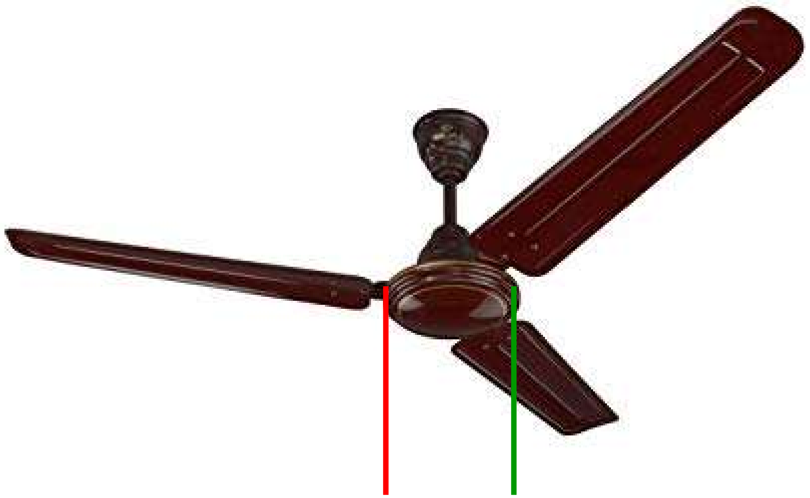
For Help, click Help Topics on the Help Menu.

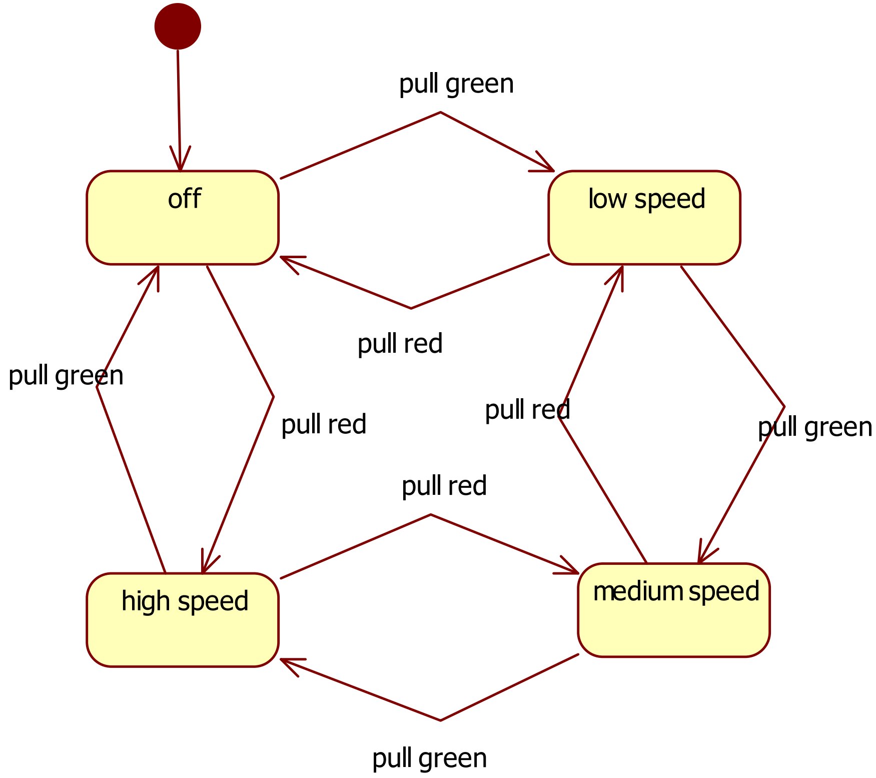
492,342

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.

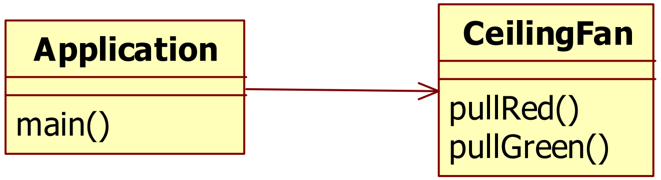
Ceiling fan




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

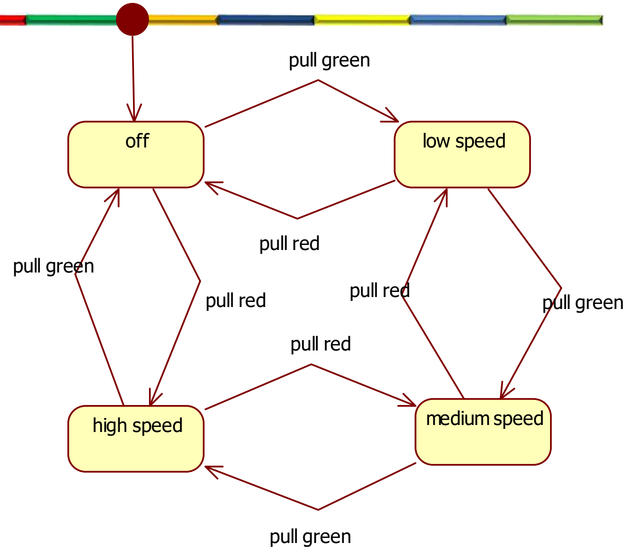
```
public class Application {
  public static void main(String[] args) {
    CeilingFan fan = new CeilingFan();
    fan.pullgreen();
    fan.pullgreen();
    fan.pullred();
    fan.pullred();
  }
}
```



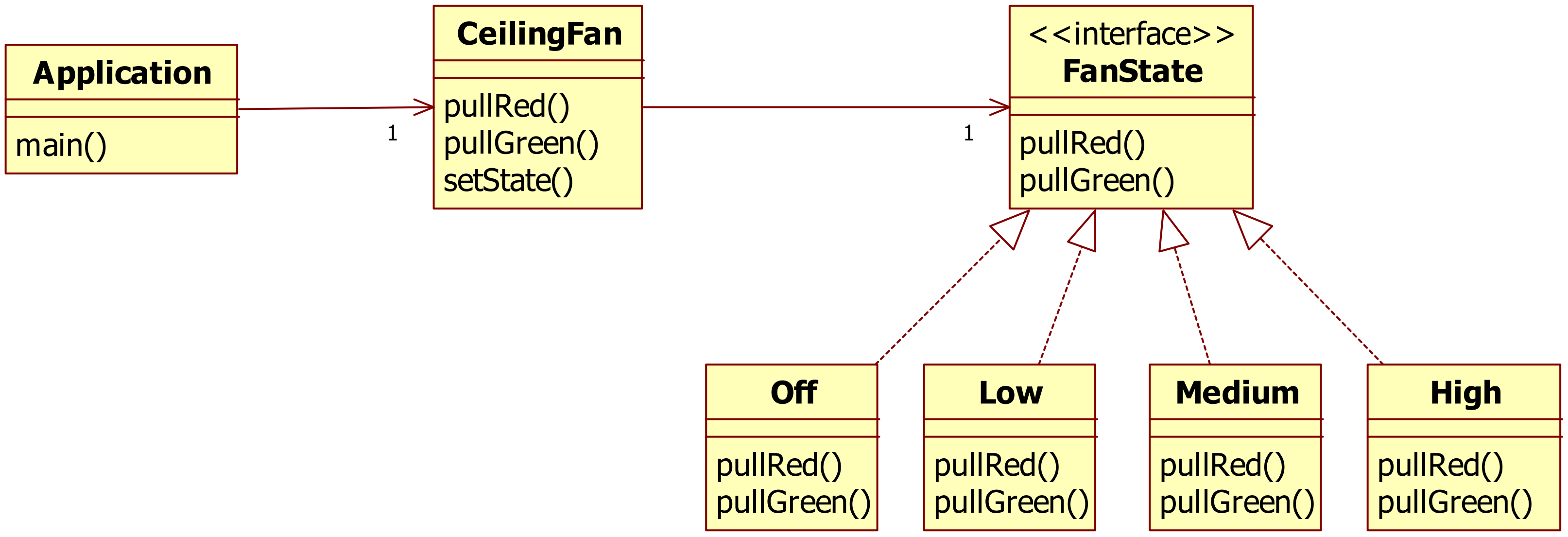
```
public class CeilingFan {
  int current_state = 0;

  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



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



Application
main()

1

CeilingFan
pullRed()
pullGreen()
setState()

1

<<interface>>
FanState
pullRed()
pullGreen()

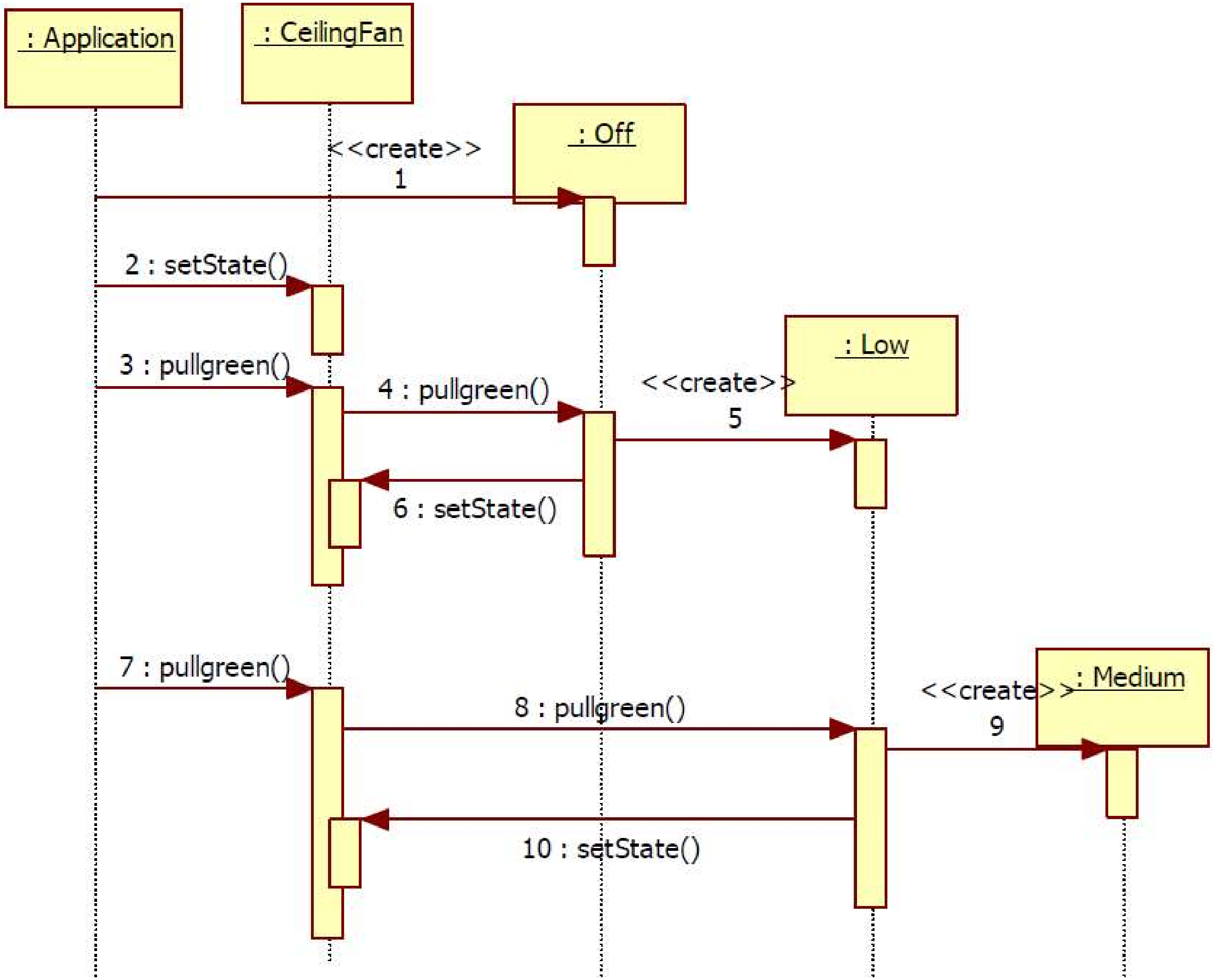
Off
pullRed()
pullGreen()

Low
pullRed()
pullGreen()

Medium
pullRed()
pullGreen()

High
pullRed()
pullGreen()

With state


: Application
: CeilingFan
: Off
<<create>>
1
2 : setState()
3 : pullgreen()
4 : pullgreen()
: Low
<<create>>
5
6 : setState()
7 : pullgreen()
8 : pullgreen()
<<create>>
9
: Medium
10 : setState()

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

```java
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);
  }
}
```

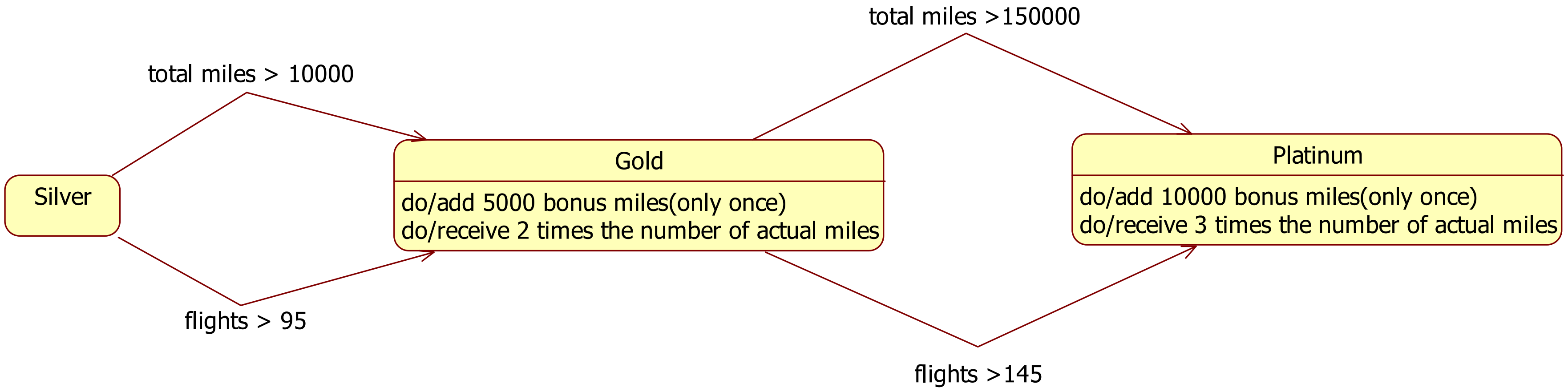
```java
public class Low implements FanState{
  CeilingFan fan;

  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);
  }
}
```

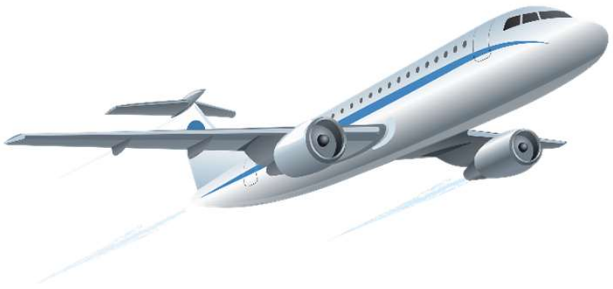
With state

```java
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);
  }
}
```

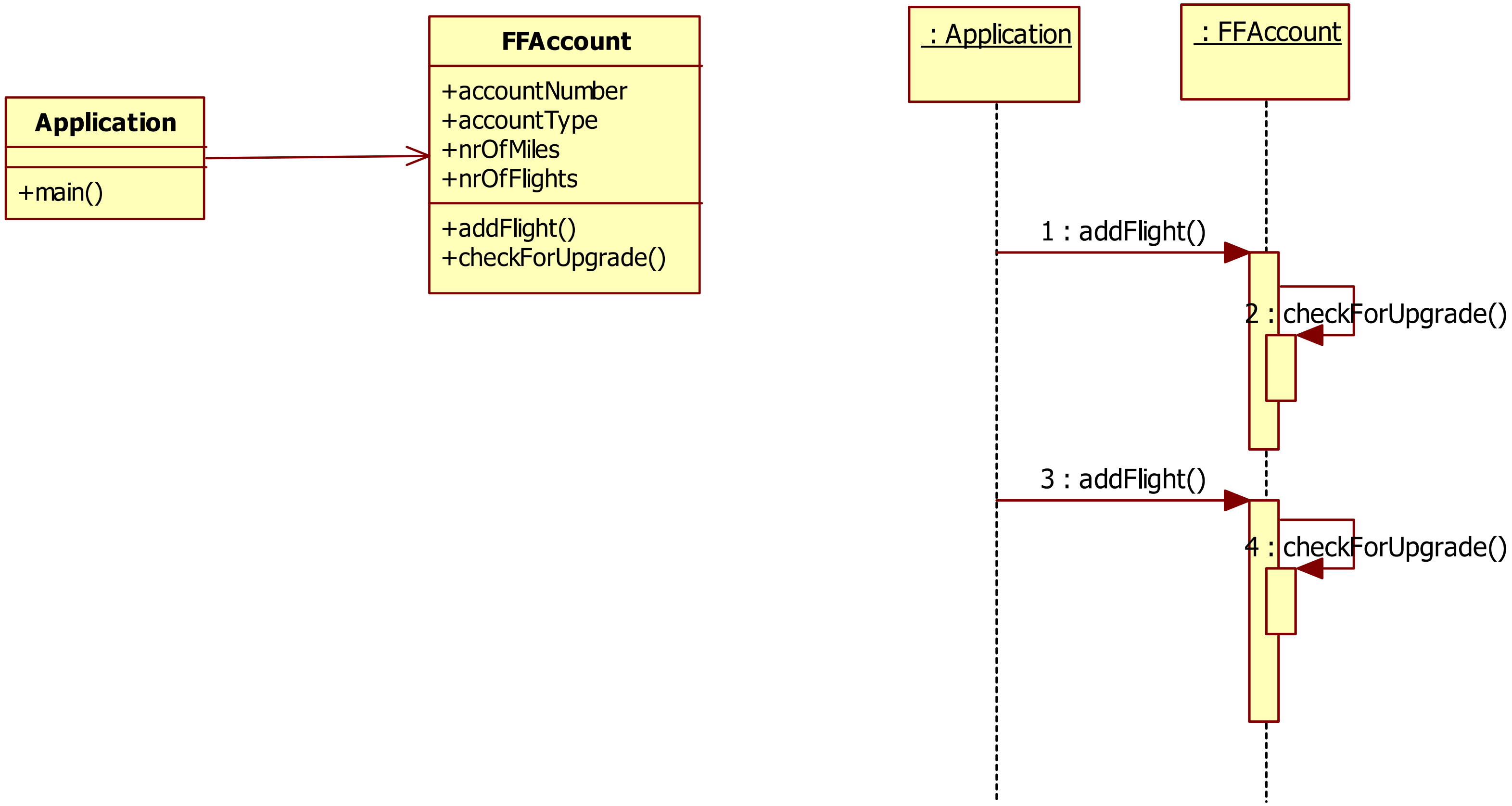
```java
public class High implements FanState{
  CeilingFan fan;

  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


total miles > 10000
flights > 95
Silver
Gold
do/add 5000 bonus miles(only once)
do/receive 2 times the number of actual miles
total miles >150000
flights >145
Platinum
do/add 10000 bonus miles(only once)
do/receive 3 times the number of actual miles



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

```java
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

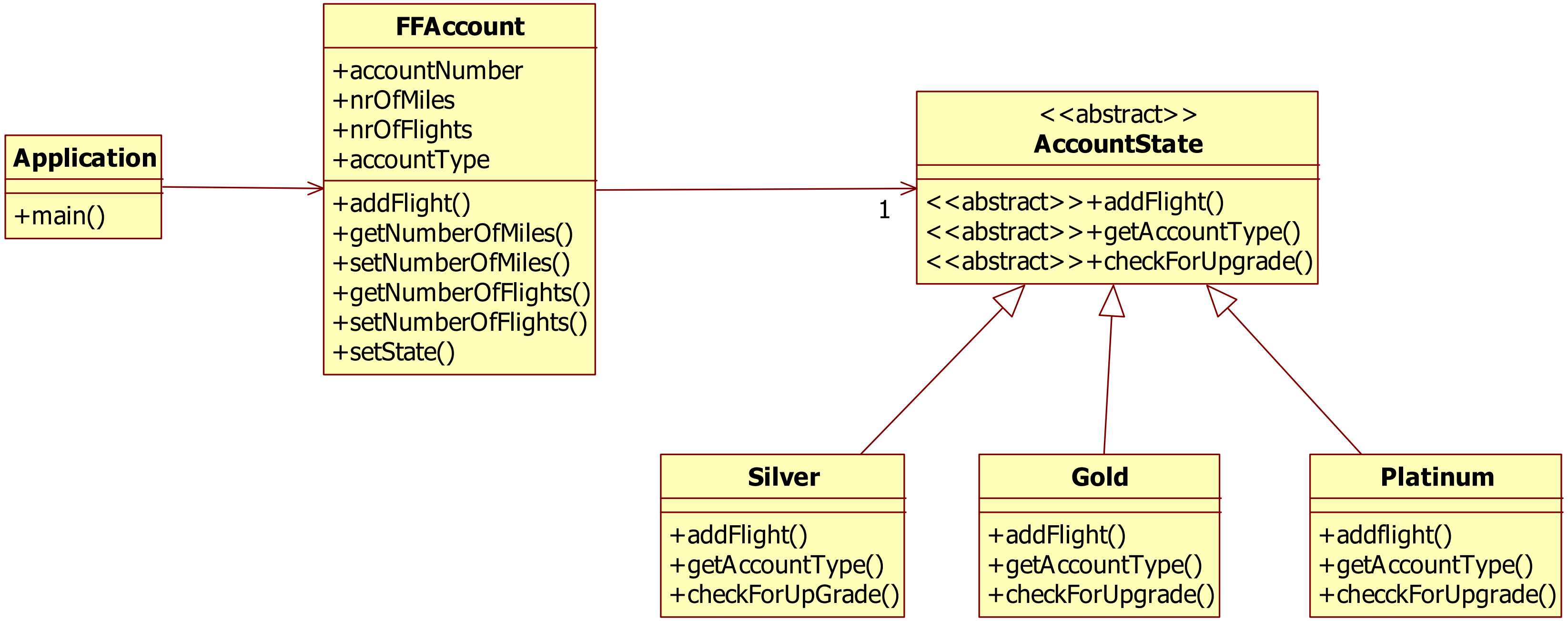
```
public class FFAccount {
...
public void addFlight(int newMiles) {
  if (accountType.equals("silver")) {
    numberOfMiles += newMiles;
    numberOfFlights++;
    checkForUpgrade();
  } else {
    if (accountType.equals("gold")) {
      numberOfMiles += (2 * newMiles);
      numberOfFlights++;
 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

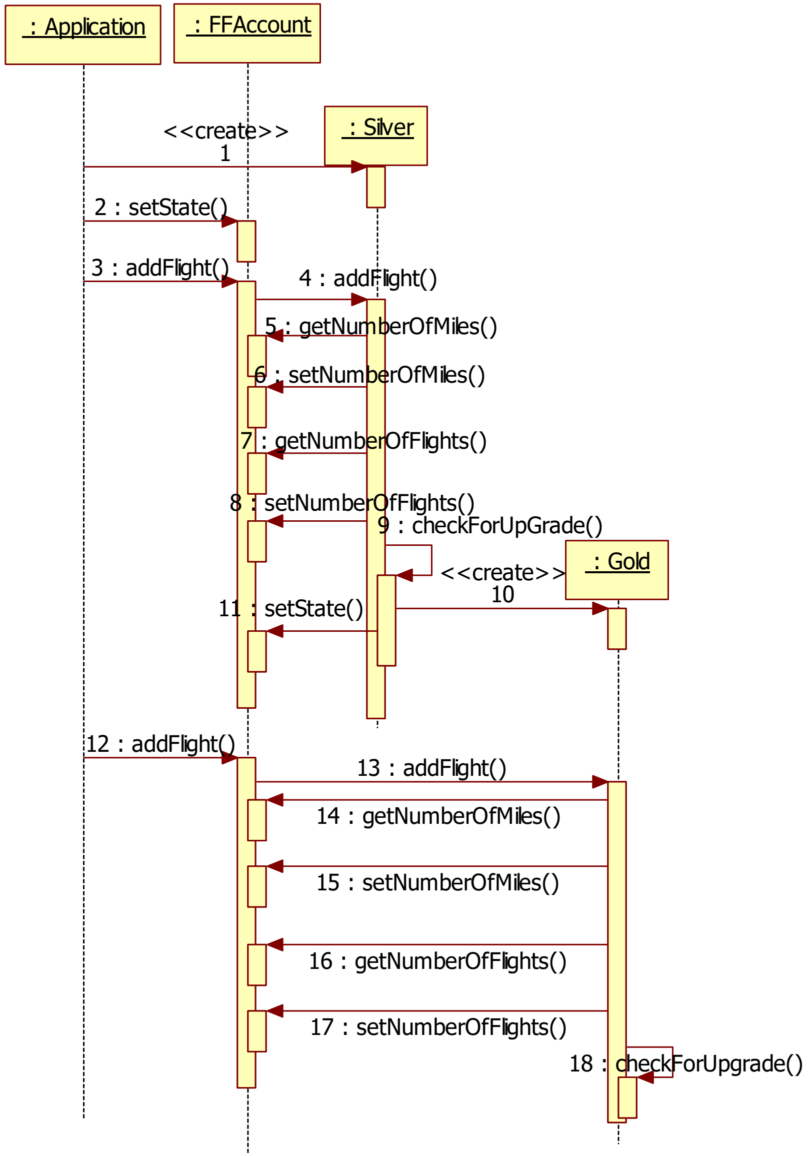
Complex if-then-else logic

Logic of state is not reusable for other classes

With state


FFAccount
+accountNumber
+nrOfMiles
+nrOfFlights
+accountType
+addFlight()
+getNumberOfMiles()
+setNumberOfMiles()
+getNumberOfFlights()
+setNumberOfFlights()
+setState()
Application
+main()
1
<<abstract>>
AccountState
<<abstract>>+addFlight()
<<abstract>>+getAccountType()
<<abstract>>+checkForUpgrade()
Silver
+addFlight()
+getAccountType()
+checkForUpGrade()
Gold
+addFlight()
+getAccountType()
+checkForUpgrade()
Platinum
+addflight()
+getAccountType()
+checcckForUpgrade()

With state


: Application
: FFAccount
<<create>>
1
: Silver
2 : setState()
3 : addFlight()
4 : addFlight()
5 : getNumberOfMiles()
6 : setNumberOfMiles()
7 : getNumberOfFlights()
8 : setNumberOfFlights()
9 : checkForUpGrade()
<<create>>
10
: Gold
11 : setState()
12 : addFlight()
13 : addFlight()
14 : getNumberOfMiles()
15 : setNumberOfMiles()
16 : getNumberOfFlights()
17 : setNumberOfFlights()
18 : checkForUpgrade()

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);
      account.setNumberOfMiles(account.getNumberOfMiles()+5000);
    }
  }

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

Gold state

```java
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);
      account.setNumberOfMiles(account.getNumberOfMiles()+10000);
    }
  }

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

Platinum state

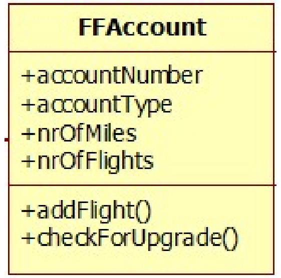
```java
public class Platinum extends AccountState {
  public Platinum(FFAccount account) {
    super(account);
  }

  public void addFlight(int newMiles) {
    account.setNumberOfMiles(account.getNumberOfMiles() + (3 * newMiles));
    account.setNumberOfFlights(account.getNumberOfFlights() + 1);
  }

  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 =117000
```

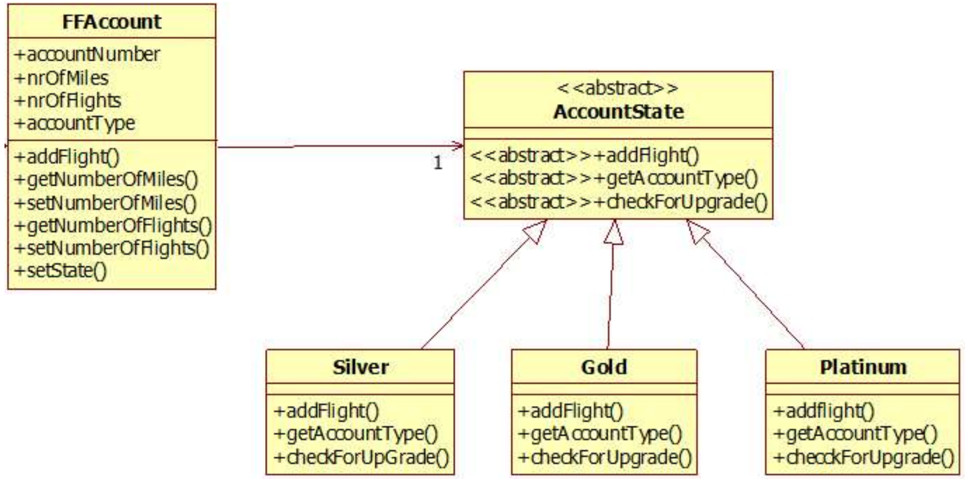
State advantages



Complex if-then-else logic

Difficult to add new state

Difficult to change state logic



Simpler if-then-else logic

Easier to add new state

Easier to change state logic

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.