State pattern

Consider the following simple Frequent Flyer Miles Application:

```java
public class Application {
  public static void main(String[] args) {
      FFAccount ffaccount =  new FFAccount("213425");
      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());
  }
}


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

      public FFAccount(String aNumber) {
            accountNumber=aNumber;
            accountType="silver";
      }

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

      public void checkForUpgrade(){
            if ((numberOfMiles > 100000)|| (numberOfFlights > 95)){
                  accountType ="gold" ;
            }
      }

      public String getAccountType() {return accountType;}
      public void setAccountType(String accountType)
         {this.accountType = accountType;}
      public int getNumberOfMiles() {return numberOfMiles;}
      public int getNumberOfFlights() {return numberOfFlights;}
      public String getAccountNumber() {return accountNumber;}
      public void setAccountNumber(String accountNumber)
        {this.accountNumber = accountNumber;}
}
```

The business rules for this application are:

- There are 2 types of accounts, “silver” and “gold”.
- Everyone starts with a “silver” account.
- When you have more than 100.000 miles or more than 95 flights, you are upgraded to a “gold” account.
- Silver accounts receive the same number of miles as the actual miles of their flights.
- Gold accounts receive 2 times the number of miles as the actual miles of their flights.

This application has the following problems:

- The FFAccount class needs to change every time we change the business rules. If we add a new account type “Platinum”, we have to change the FFAccount class.
- The addFlight() method of the FFAccount class becomes a long if-then-else structure which is complicated to maintain (to understand and to change)

Redesign and rewrite the code of this application when we change the business rules as follows:

- There are 3 types of accounts, “silver”, “gold” and “platinum”.
- Everyone starts with a “silver” account.
- When you have more than 75.000 miles or more than 75 flights, you are upgraded to a “gold” account.
- When you have more than 150.000 miles or more than 145 flights, you are upgraded to a “platinum” account.
- Silver accounts receive the same number of miles as the actual miles of their flights.
- Gold accounts receive 2 times the number of miles as the actual miles of their flights.
- Platinum accounts receive 3 times the number of miles as the actual miles of their flights.
- Platinum accounts receive 2 times the number of flights as the actual number of flights.

Your solution should reflect the following requirements:

1. It should be easy to add new account types, and change the given business rules without changing the FFAccount class at all.
2. Nowhere in this application we want to see an if-then-else construct

Draw the class diagram with the State pattern applied. Draw a sequence diagram that shows how the State pattern works.