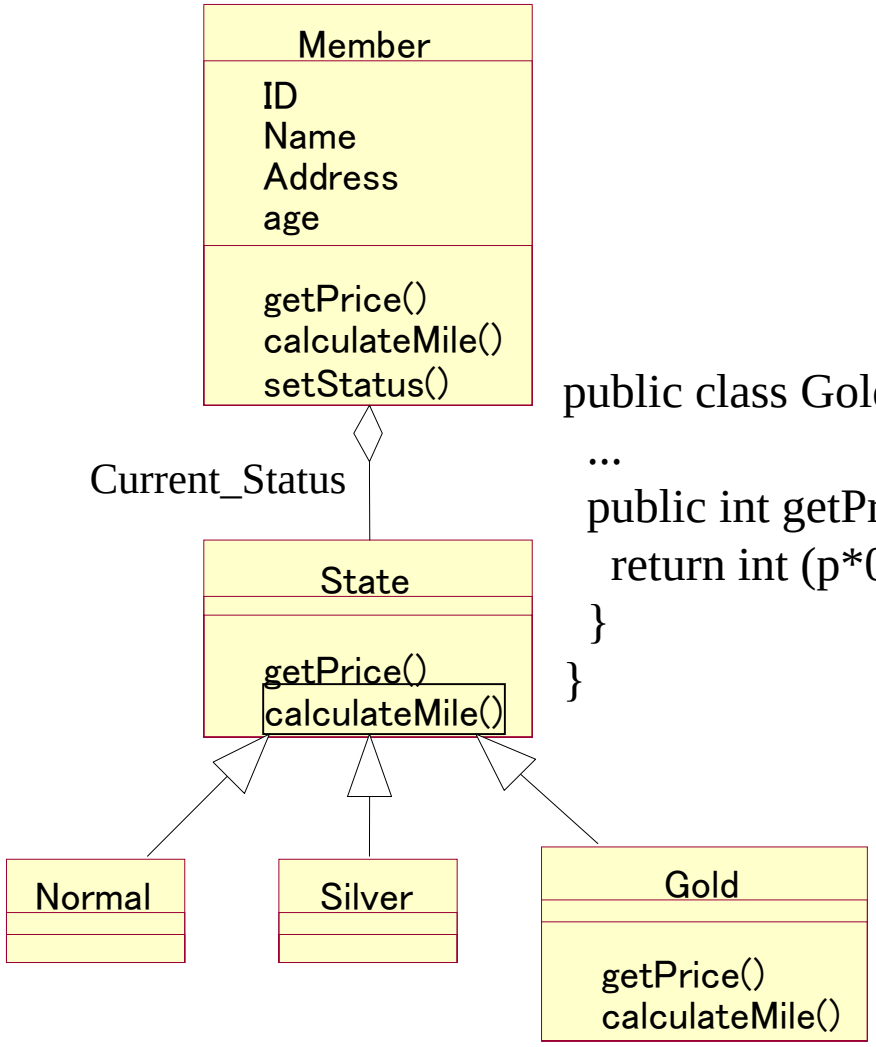
Design Pattern

	生成	構造	振舞い
クラス	Factory Method	Adapter Bridge	Template Method
オブジェクト	Abstract Factory Prototype Solitaire	Adapter Bridge Flyweight Glue Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy
複合	Builder	Composite Wrapper	Interpreter Iterator Visitor

マイレージプログラム（State Pattern 使用）

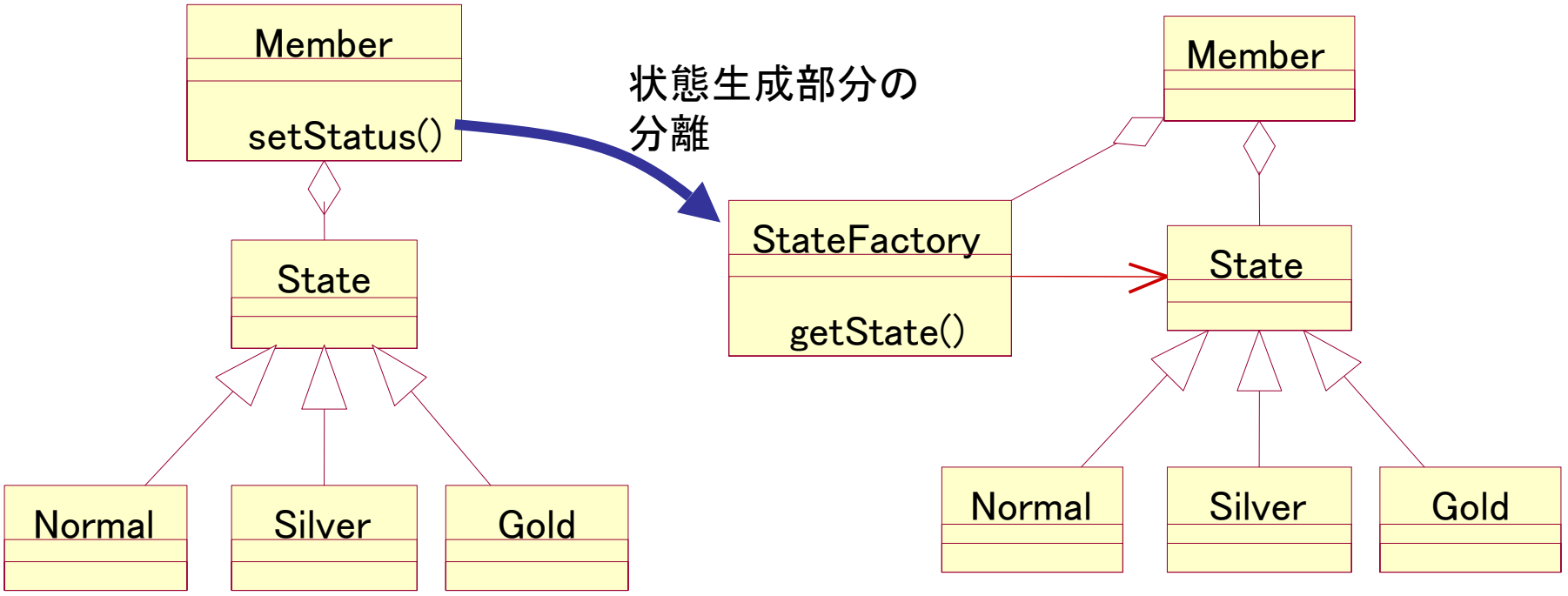



```
public class Gold {
  ...
  public int getPrice(int p) {
    return int (p*0.95)
  }
}
```

```
public class Member {
   state Current_Status ;
   ...

  public Member(int tm) {
     setStatus(tm) ;
   }

  public void setStatus(int tm) {
     if (tm < 25000) {
        Current_Status = new Normal(); }
     else if (tm < 50000) {
        Current_status = new Silver() ;}
     else if (tm < 75000) {
        Current_Status = new Gold() ;} ;
   }

  public int getPrice(int p) {
      return Current_Status.getPrice(p) ;
   }
  ...
}
```

メンバーの種別が増えたとき
大きくなる.

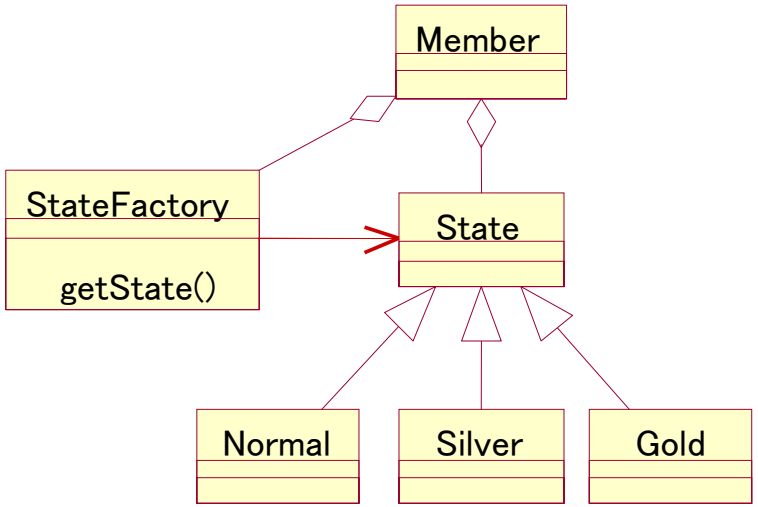
オブジェクト生成部分の分離



State Pattern

State Pattern + Simple Factory

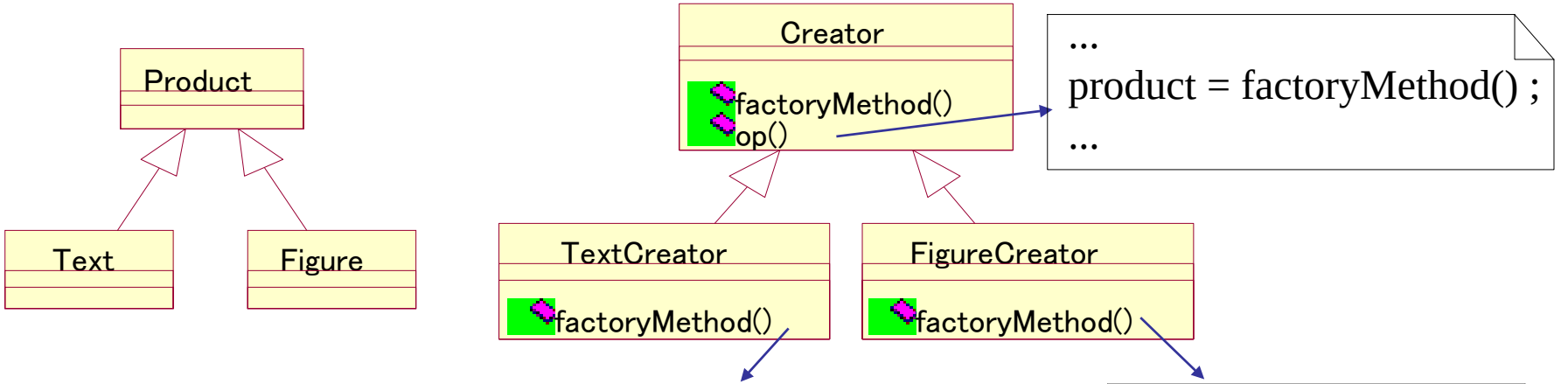
Simple Factory Pattern




```
public class Member {
    state Current_Status ;
    ...

   static public void main(String[] argv) {
      StateFactory SF = new StateFactory();
      ...

   public Member(int tm) {
       Current_Status = SF.getState(tm) ;
    }

   public int getPrice(int p) {
        return Current_Status.getPrice(p) ;
    }
     ...
}
```

Factory Method (1)

インスタンスの生成(Constructor)のインターフェースを統一
異なるクラスのインスタンスに同じような処理を施す，
抽象クラスしかわかっていないときに必要

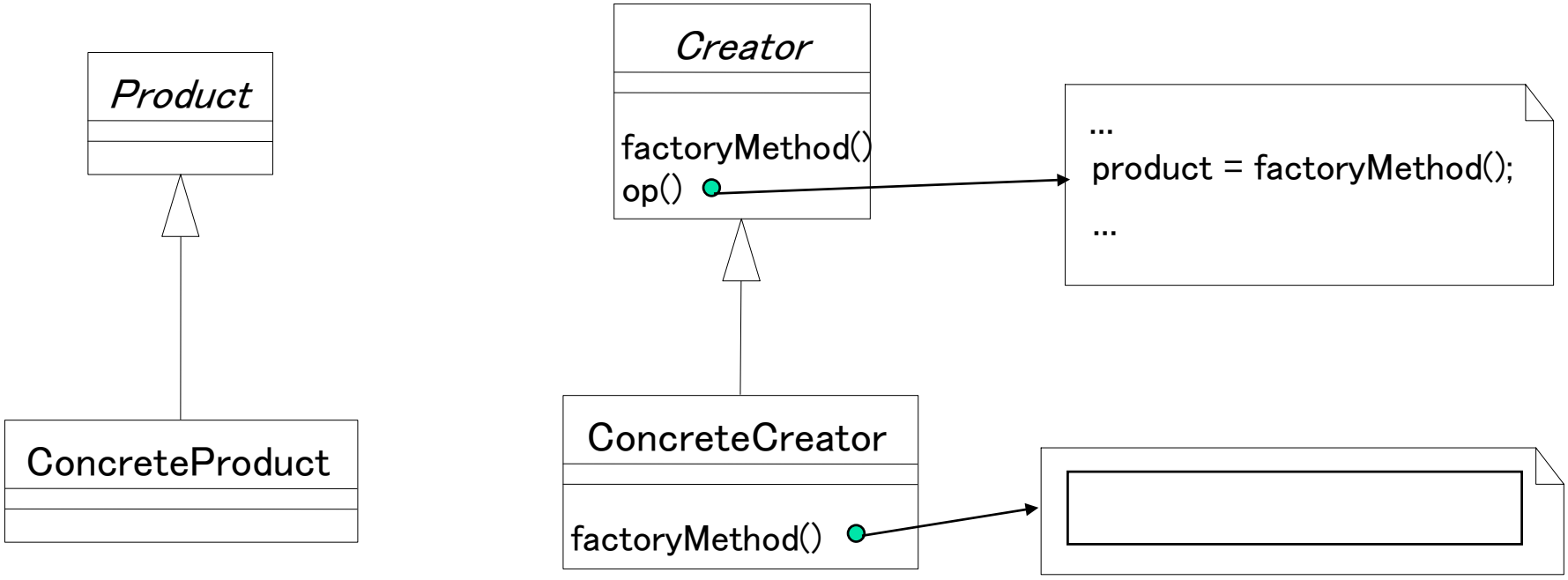



```
c1 = new TextCreator(12) ;
c1.op() ;
...
c2 = new FigureCreator(0,0) ;
c2.op() ;
...
```

```
public TextCreator(point) {
 ...
}
```

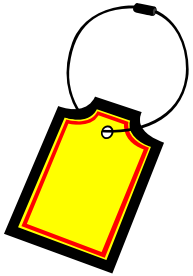
```
public FigureCreator(x,y) {
 ...
}
```

Factory Method (2)



Abstract Factory (1)

マイレージプログラムで，メンバのNormal, Silver, Gold のグレードに応じて以下のような景品がでるとする．
設計は？




Normal : 会員証
Silver : 会員証，荷物につけるタグ（氏名，住所入り）
Gold : 会員証，荷物につけるタグ，
　　日本国内往復航空券（区間は会員指定可）

Abstract Factory (2)