Технология программирования

Шаблоны проектирования

Обязанности классов

Обязанность (responsibility) – это контракт или обязательство класса:

- ***Обязанность знания (knowing)***. Реализуется методами доступа (не изменяется состояние объекта). Объект предоставляет:
  - информацию о закрытых инкапсулированных данных
  - информацию о связанных (агрегируемых) объектах
  - результаты вычислений
- ***Обязанность действия (doing)***. Реализуется методами управления и реализации (изменяют состояние)
  - выполнение действий над самими объектами
  - инициация действий, выполняемых другими объектами
  - координация других объектов

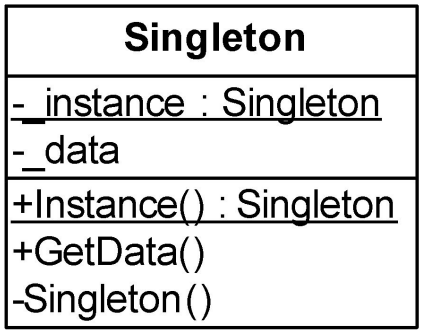
Шаблон проектирования (design pattern) – это готовое решение для часто встречающихся задач. Шаблон описывает распределение обязанностей классов и объектов в определенной ситуации.

- ***Порождающие шаблоны.*** Описывают ЧТО создается, КЕМ создается, КАК создается и КОГДА создается. Локализуют сведения об особенностях создаваемого объекта и процедуре создания.
- ***Структурные шаблоны.*** Описывают как составлять сложные структуры из классов и объектов с определенными свойствами.
- ***Шаблоны поведения.*** Описывают каким образом можно организовать и изменять поведение объектов

«Одиночка» (Singleton)

Гарантирует, что у класса есть только один экземпляр, и предоставляет к нему доступ

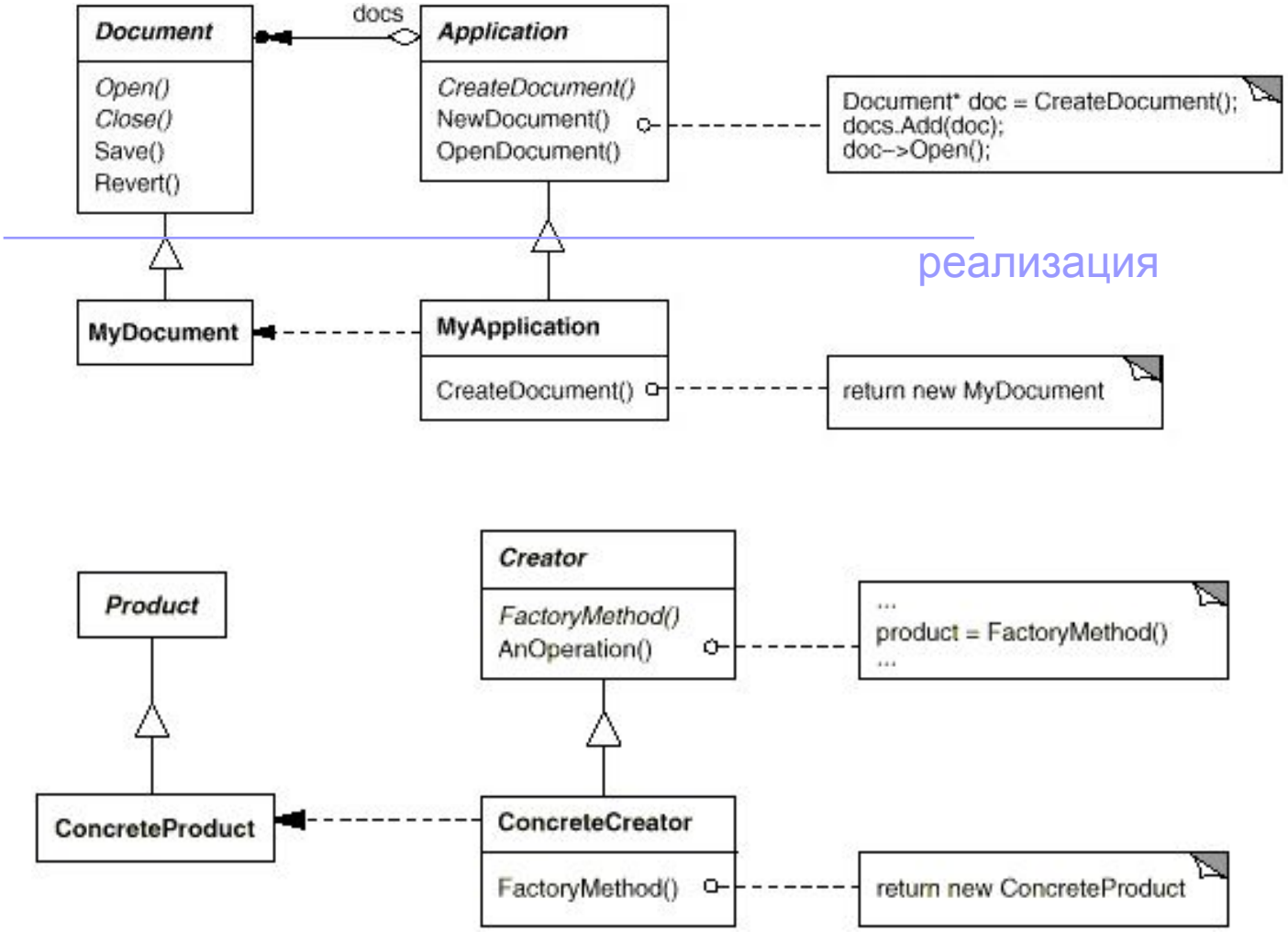
```
class Singleton
{
public:
  static Singleton* Instance()
  {
    if(_instance == 0)
    {
      _instance = new Singleton;
    }
    return _instance;
  }
  Data GetData()
  {
    return _data;
  }
private:
  Singleton() { _data.x = 3; /* действия */}
  static Singleton* _instance;
  Data data;
}
```




```
void main()
{
  Data d1 =
      Singleton::Instance()->GetData();
}
```

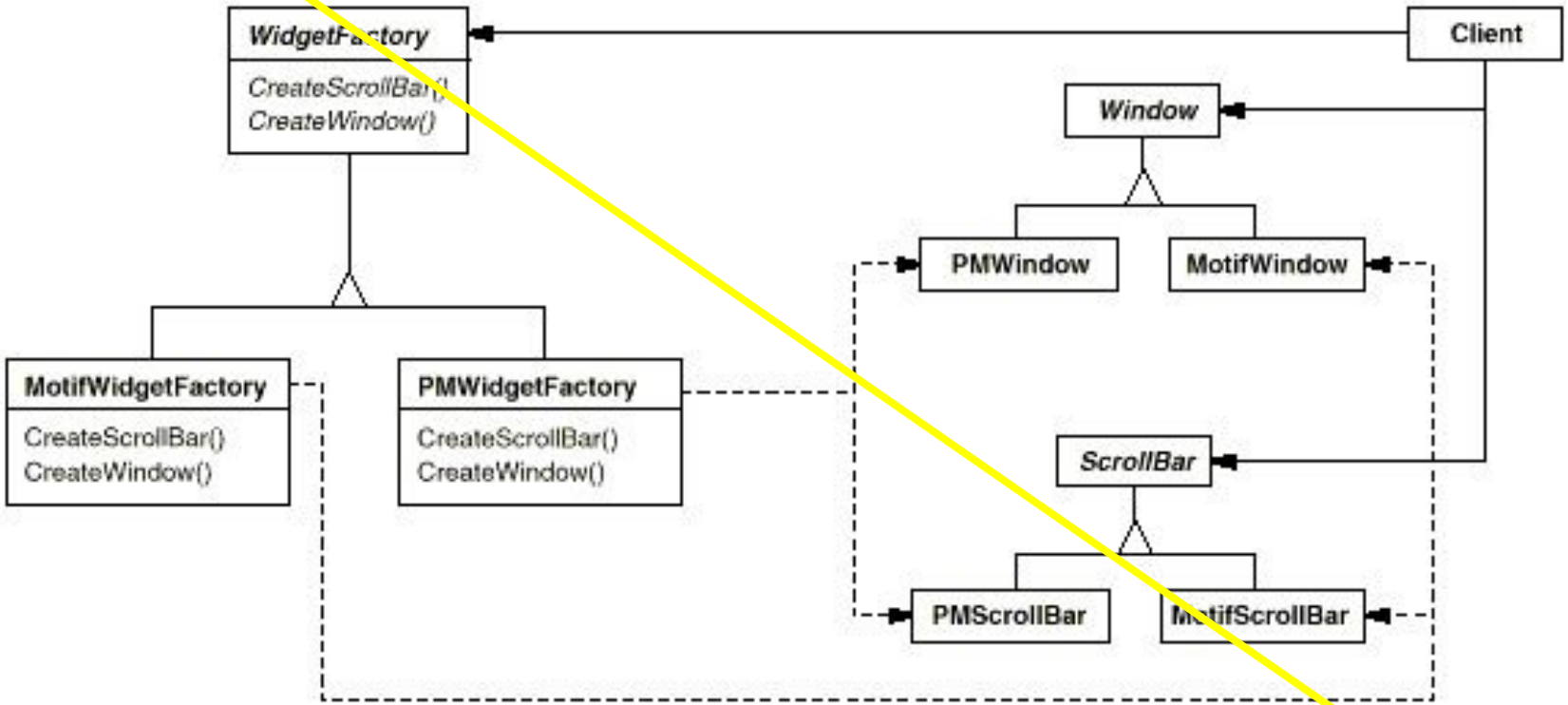
«Фабричный метод» (Factory Method)

Определяет интерфейс для создания объекта, позволяя подклассам определять, какому классу должен принадлежать создаваемый объект

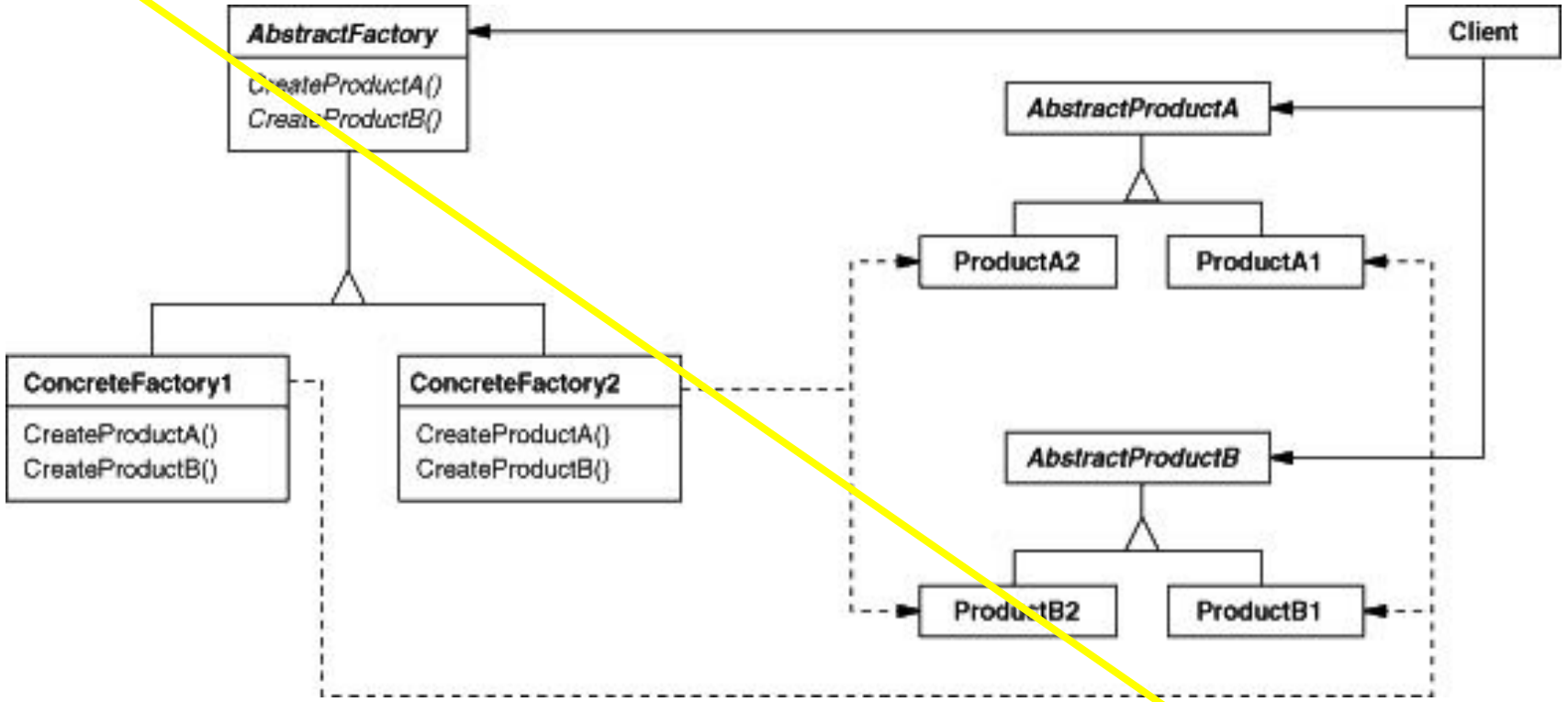


«Абстрактная фабрика» (Abstract Factory)

Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретные классы



«Абстрактная фабрика» (Abstract Factory)

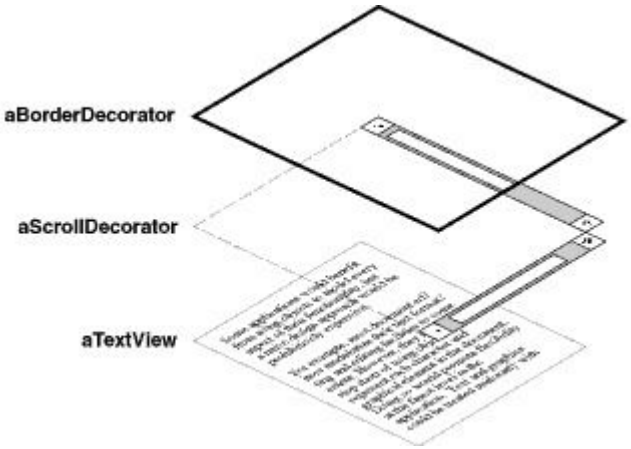


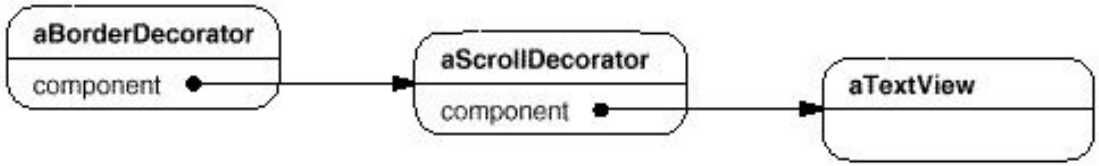
«Абстрактная фабрика» (Abstract Factory)

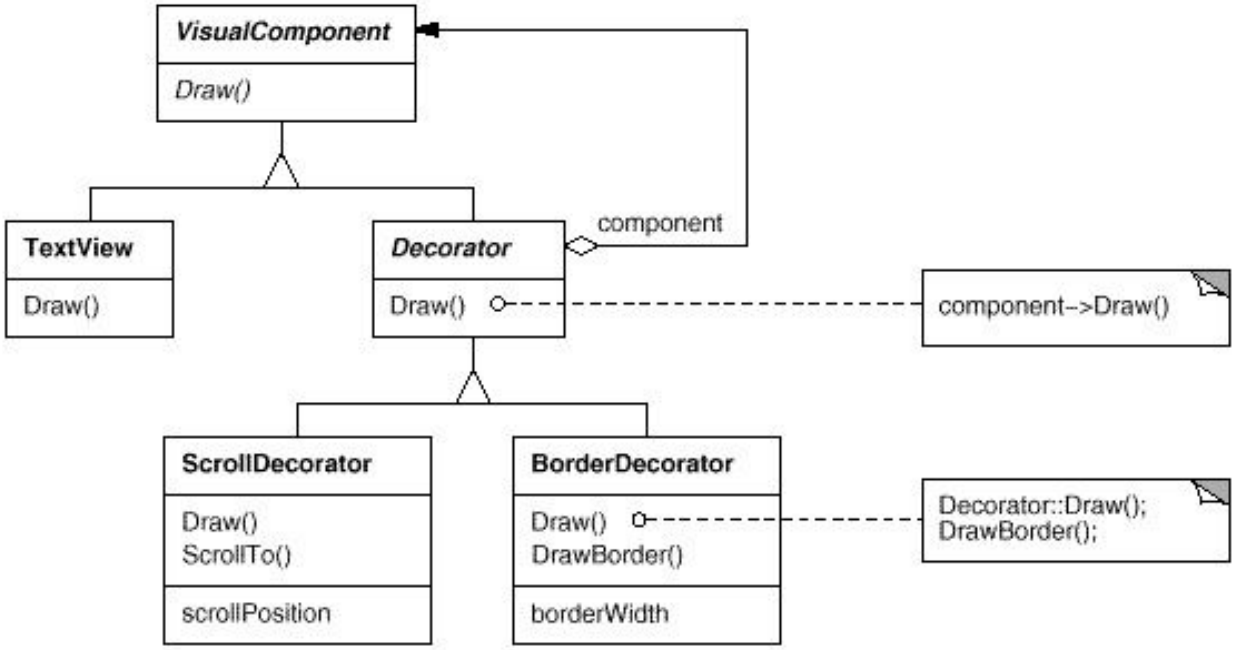
```
class Client
{
  void f(AbstractFactory* factory)
  {
    AbstractProductA aProduct = factory-> CreateProductA();
    AbstractProductB bProduct = factory-> CreateProductB();
/* Действия с aProduct, bProduct */
  }
}

void main()
{
  Client client;
  AbstractFactory* factory =
      new ConcreteFactory1(); // а можно ConcreteFactory2
  client->f(factory);
}
```

«Декоратор» (Decorator)

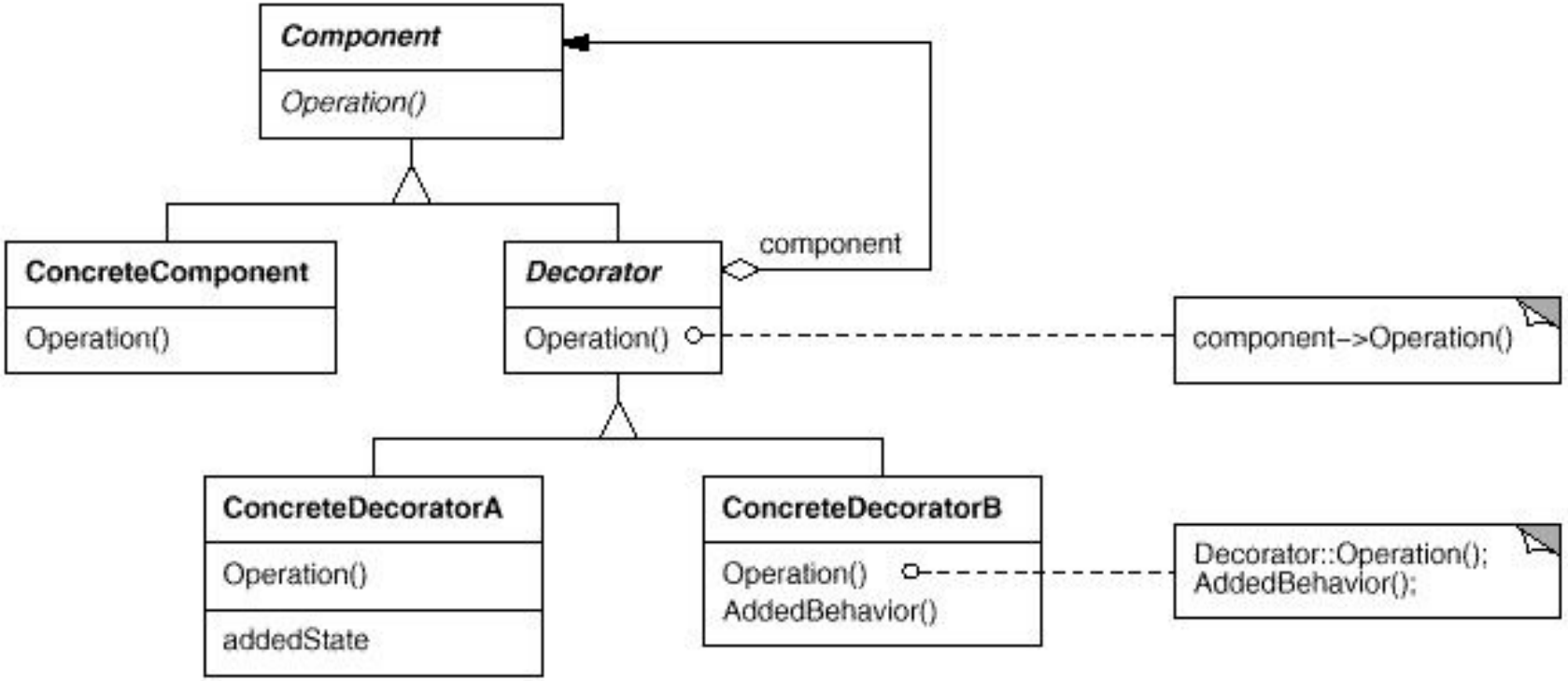
Динамически добавляет объекту новые обязанности. Является альтернативой наследованию





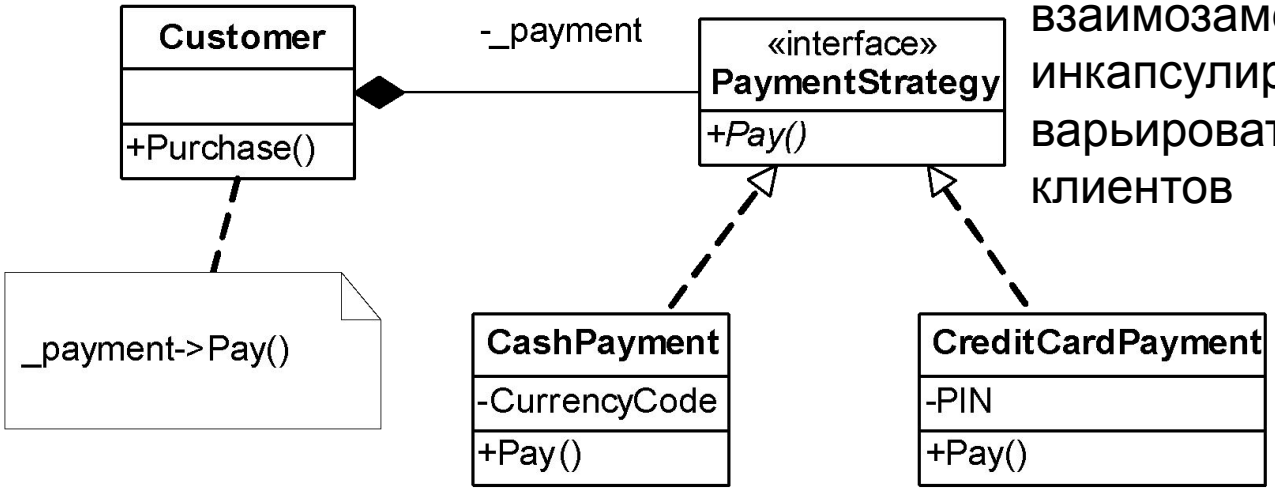


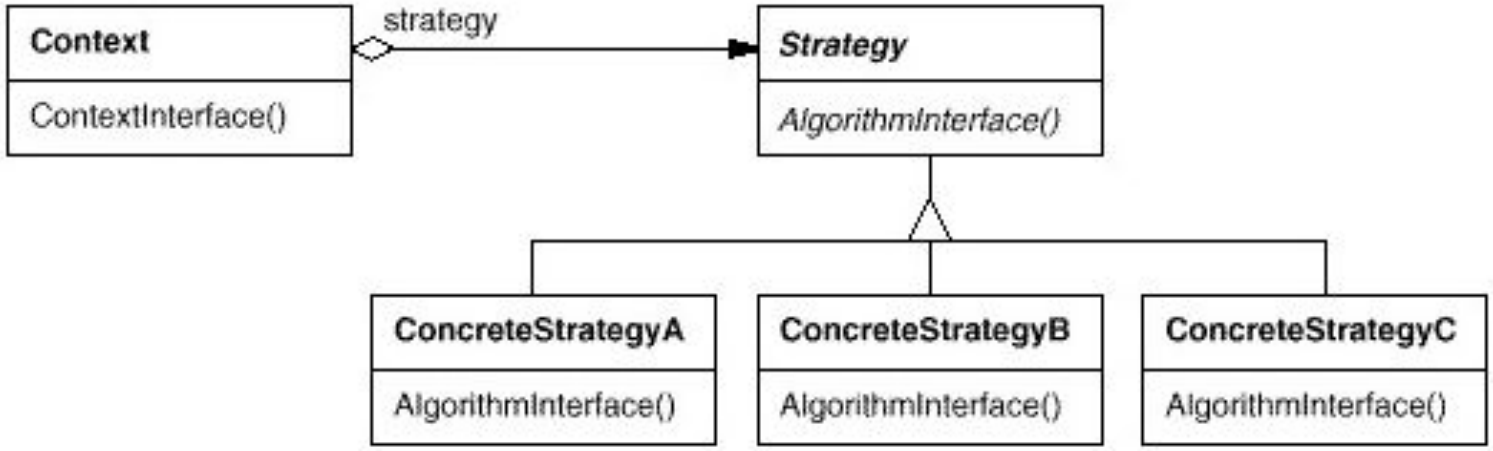
«Декоратор» (Decorator)



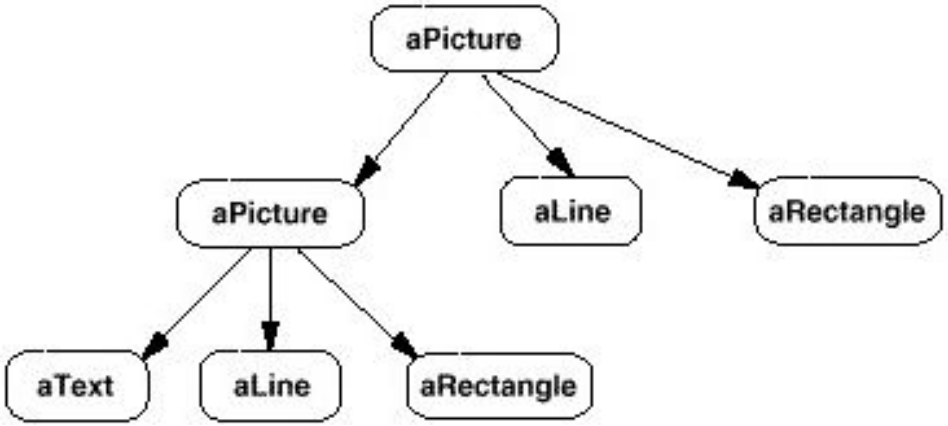
«Стратегия» (Strategy)

Определяет семейство взаимозаменяемых алгоритмов, инкапсулируя их. Алгоритмы могут варьироваться не зависимо от клиентов

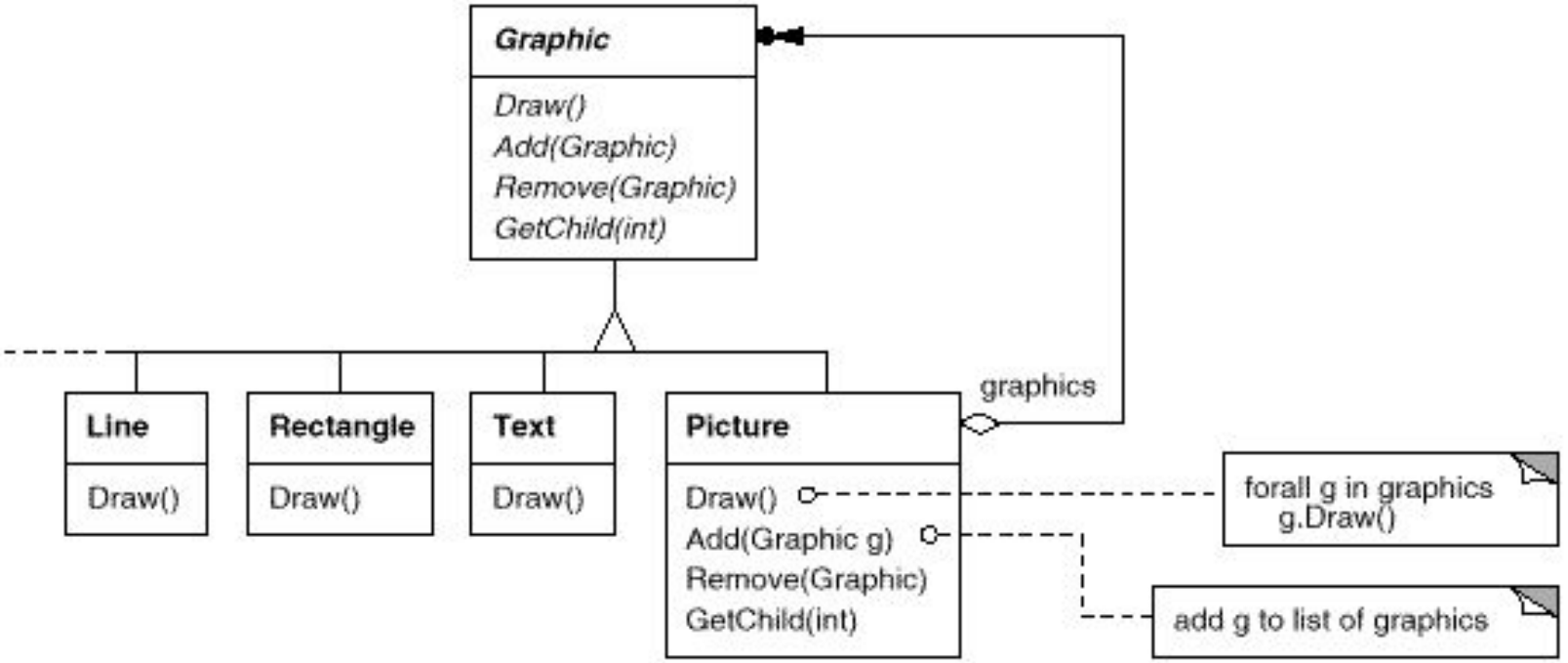




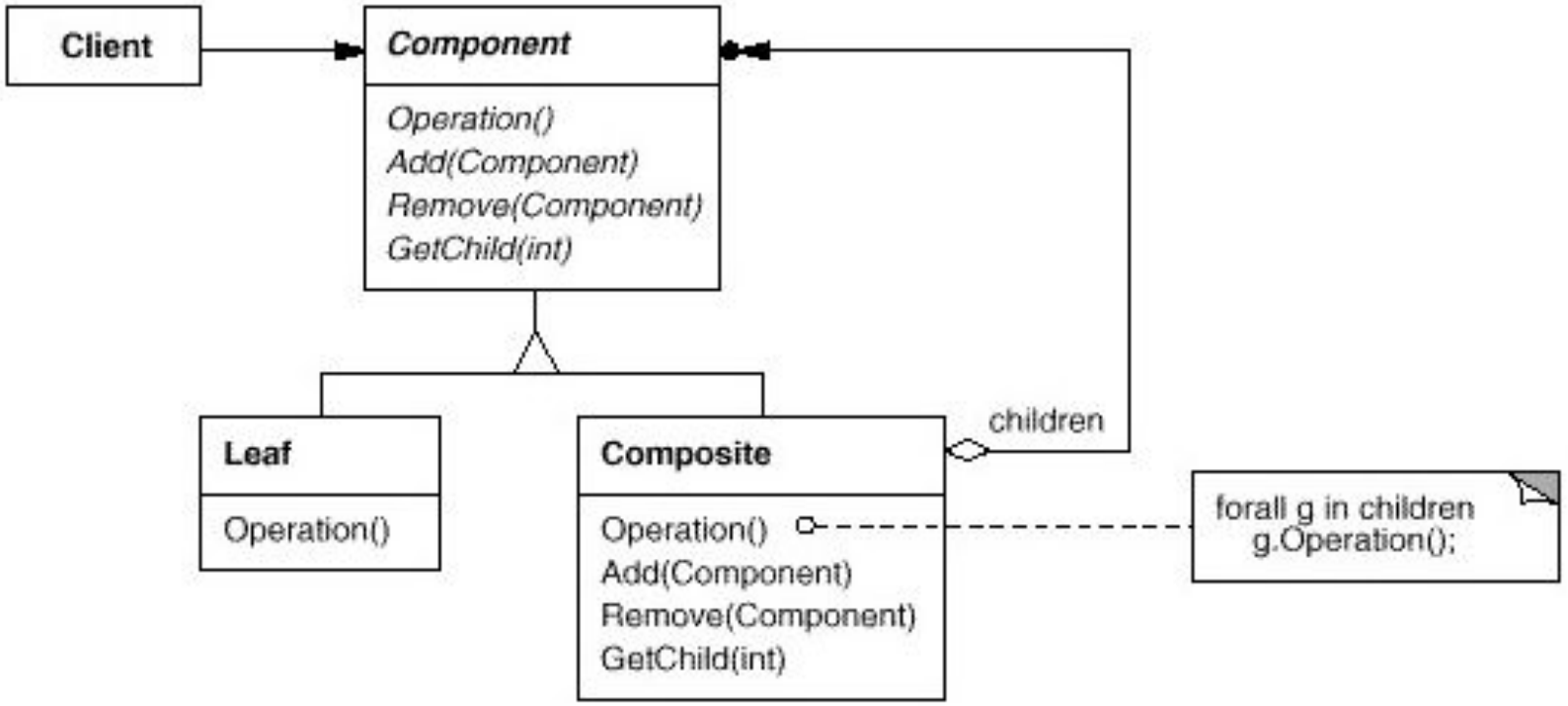
«Компоновщик» (Composite)



Компонует объекты в древовидные структуры. Позволяет клиенту единообразно взаимодействовать с простыми и составными объектами

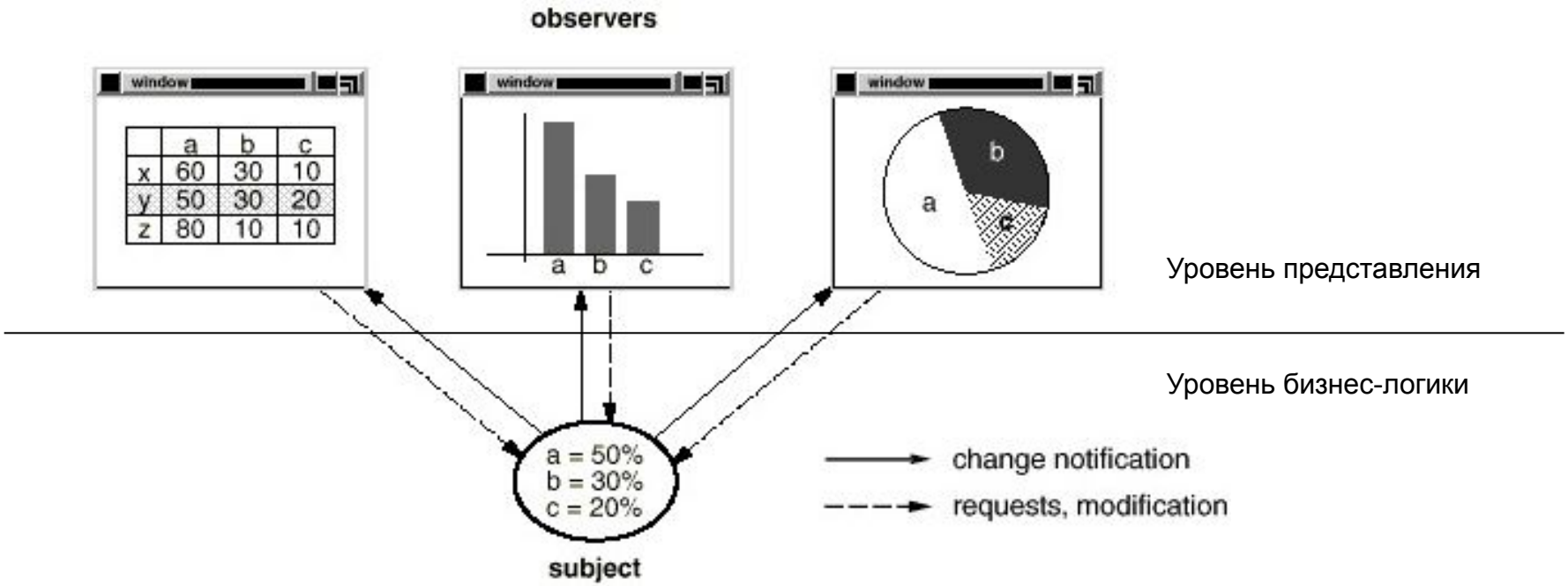


«Компоновщик» (Composite)



«Наблюдатель» (Observer)

Устанавливает отношение между объектами «один ко многим» так, чтобы при изменении состояния одного объекта другие зависимые объекты могли обновить свое состояние



«Наблюдатель» (Observer)