

1. Абстрактная фабрика

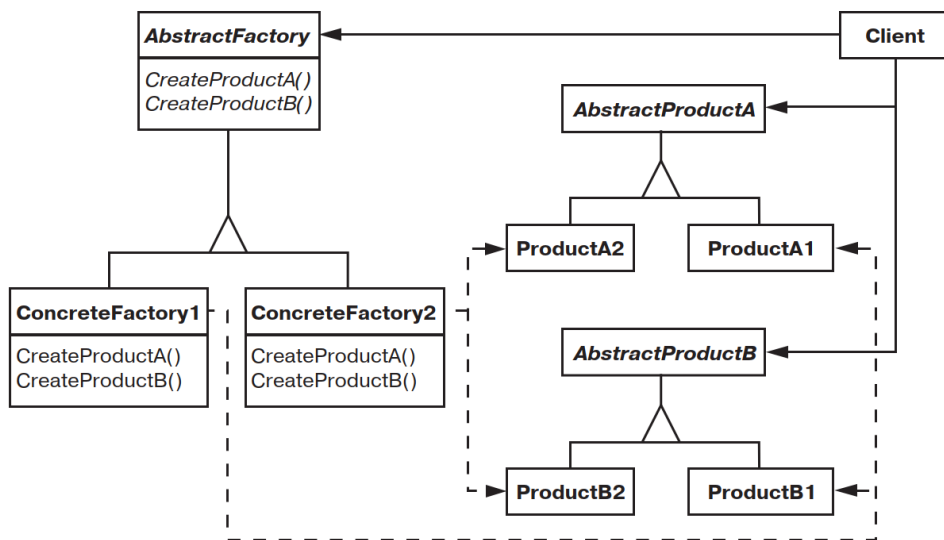
1.1 Название и классификация паттерна

Абстрактная фабрика — паттерн, порождающий объекты.

1.2 Назначение

Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

1.3 Структура



1.4 Участники

AbstractFactory — абстрактная фабрика:

- объявляет интерфейс для операций, создающих абстрактные объекты-продукты;

ConcreteFactory — конкретная фабрика:

- реализует операции, создающие конкретные объекты-продукты;

AbstractProduct — абстрактный продукт:

- объявляет интерфейс для типа объекта-продукта;

ConcreteProduct — конкретный продукт:

- определяет объект-продукт, создаваемый соответствующей конкретной фабрикой;
- реализует интерфейс **AbstractProduct**;

Client — клиент:

- пользуется исключительно интерфейсами, которые объявлены в классах **AbstractFactory** и **AbstractProduct**.

1.5 ЕО реализация шаблона

```
+package sandbox
+alias stdout org.eolang.io.stdout
+alias sprintf org.eolang.txt.sprintf
```

```
[type] > abstractFactory
if. > concreteFactory
eq.
  type
  "1"
  concreteFactory1
  concreteFactory2

[] > createProductA
createProductA. > @
  ^.concreteFactory
[] > createProductB
createProductB. > @
  ^.concreteFactory

[] > concreteFactory1
[] > createProductA
1 > @
[] > createProductB
2 > @

[] > concreteFactory2
[] > createProductA
"one" > @
[] > createProductB
"two" > @

[args...] > appAbstractFactory
abstractFactory > objFactory
args.get 0
stdout > @
sprintf
  "ProductA: %s\nProductB: %s\n"
  objFactory.createProductA
  objFactory.createProductB
```

Вывод программы

```
$ ./run.sh 1
ProductA: 1
ProductB: 2
$ ./run.sh 2
ProductA: one
ProductB: two
```

Данная программа создает объекты целые числа или строки в зависимости от параметра `args[0]`. Если `args[0] = 1`, то создадутся объекты 1 и 2, иначе – “one” и “two”.

Шаблон предполагает использования интерфейсов, который отсутствуют в ЕО. В данном случае предпринята попытка реализации интерфейса через ЕО объект имеет параметр `type` в зависимости от которого выбирается конкретная реализация фабрики объектов. Это делает зависимым объект-интерфейс, от набора реализаций этого интерфейса(при добавлении новой реализации необходимо внести изменения в объект интерфейс).

2. Singleton (одиночка)

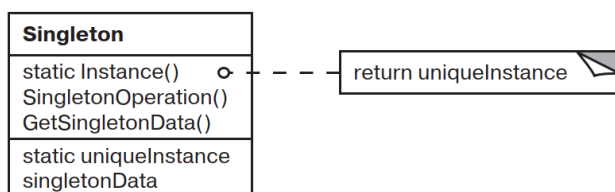
2.1 Название и классификация паттерна

Одиночка — паттерн, порождающий объекты

2.2 Назначение

Гарантирует, что у класса существует только один экземпляр, и предоставляет к нему глобальную точку доступа.

2.3 Структура



2.4 Участники

Singleton — одиночка:

- определяет операцию `Instance`, которая позволяет клиентам получить доступ к единственному экземпляру. `Instance` — это операция класса, то есть статический метод класса;
- может нести ответственность за создание собственного уникального экземпляра.

2.5 Отношения

Клиенты получают доступ к экземпляру класса `Singleton` только через его операцию `Instance`.

2.6 ЕО реализация

В ЕО отсутствуют классы поэтому этот шаблон не реализуем в чистом виде. Если же мы в терминах ЕО определим `Singleton` как объект у которого гарантировано существует только одна копия, то реализация этого объекта тоже невозможна по следующим причинам:

- В ЕО отсутствуют ссылки. Любое использование объекта в месте отличном от места определения есть копирование этого объекта.
- В ЕО отсутствуют возможности ограничения доступа к объектам и запрета его копирования. Невозможно ограничить создание копий объекта.

3. Прототип

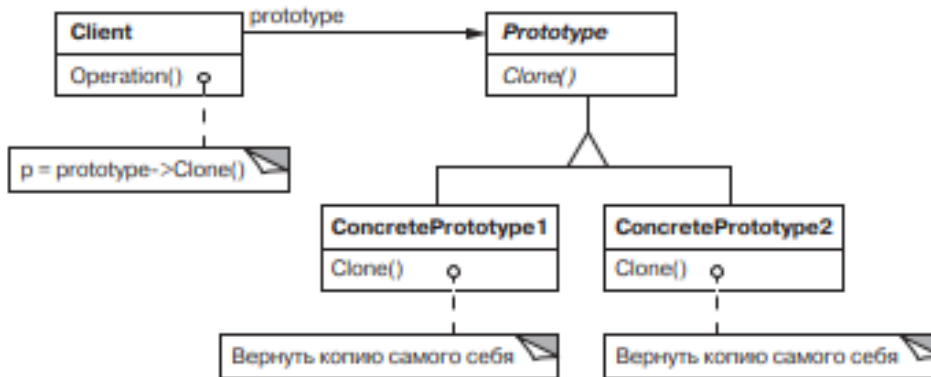
3.1 Название и классификация паттерна

Прототип — паттерн, порождающий объекты.

3.2 Назначение

Задаёт виды создаваемых объектов с помощью экземпляра-прототипа и создает новые объекты путем копирования этого прототипа.

Структура



3.3 Участники

— Prototype — прототип:

- объявляет интерфейс для клонирования самого себя;

— ConcretePrototype:

- реализует операцию клонирования себя;

— Client — клиент:

- создает новый объект, обращаясь к прототипу с запросом клонировать себя.

3.4 EO реализация

В EO каждый объект может быть скопирован, функции шаблона может выполнять каждый объект.

4. Мост(BRIDGE)

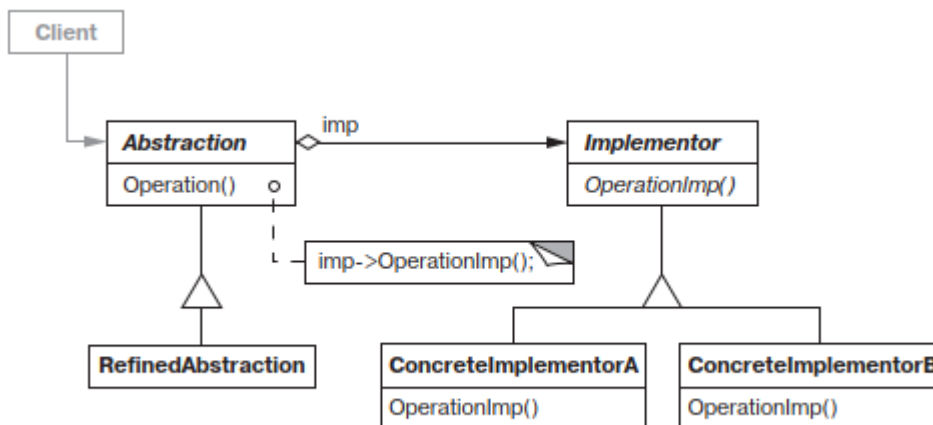
4.1 Название и классификация паттерна

Мост — паттерн, структурирующий объекты.

4.2 Назначение

Отделить абстракцию от ее реализации так, чтобы то и другое можно было изменять независимо.

Структура



4.3 Участники

Abstraction — абстракция:

- определяет интерфейс абстракции;
- хранит ссылку на объект типа Implementor;

RefinedAbstraction — уточненная абстракция:

- расширяет интерфейс, определенный абстракцией Abstraction;

Implementor — реализатор:

• определяет интерфейс для классов реализации. Он не обязан точно соответствовать интерфейсу класса Abstraction. На самом деле оба интерфейса могут быть совершенно различны. Обычно интерфейс класса Implementor предоставляет только примитивные операции, а класс Abstraction определяет операции более высокого уровня, основанные на этих примитивах;

ConcreteImplementor — конкретный реализатор:

- реализует интерфейс класса Implementor и определяет его конкретную реализацию.

4.4 Отношения

Объект Abstraction перенаправляет запросы клиента своему объекту Implementor.

4.5 ЕО реализация

```
+package sandbox
+alias stdout org.eolang.io.stdout
+alias sprintf org.eolang.txt.sprintf
```

```
[] > double
```

```
[a] > execute
```

```
  add. > @
```

```
    a
```

```
    a
```

```
[a b] > perimeter
```

```
double > doubleImpl
```

```
[] > calculate
```

```
  add. > @
```

```
    ^.doubleImpl.execute
```

```
    ^.a
```

```
    ^.doubleImpl.execute
```

```
    ^.b
```

```
[args...] > appBridge
```

```
stdout > @
```

```
  sprintf
```

```
    "%d"
```

```
  calculate.
```

```
    perimeter
```

```
    toInt.
```

```
      args.get 0
```

```
    toInt.
```

```
      args.get 1
```

Программа вычисляет периметр прямоугольника по длинам его сторон заданным в `args[0]` и `args[1]`.
Объект `perimeter` выступает в качестве `Abstraction`, объект `double – Implementor`.

5. Цепочка обязанностей (CHAIN OF RESPONSIBILITY)

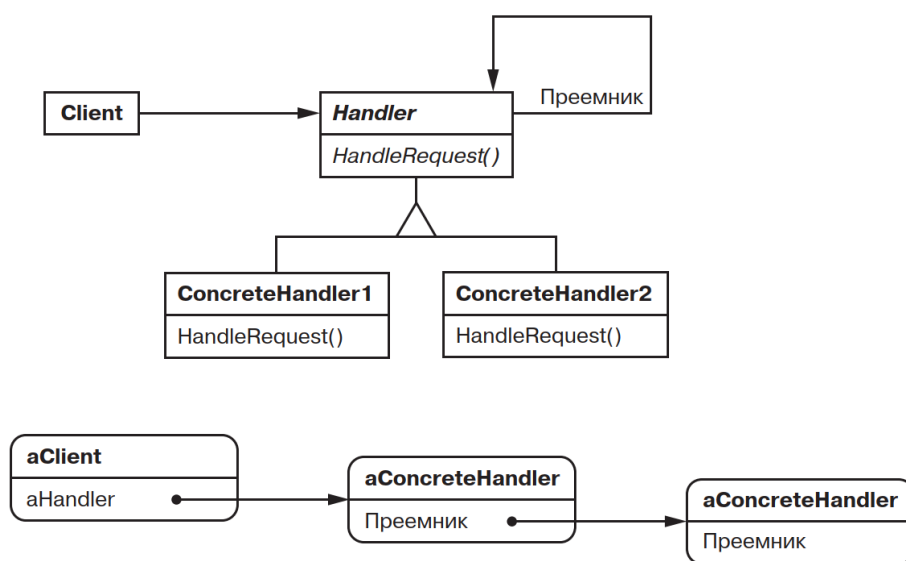
5.1 Название и классификация паттерна

Цепочка обязанностей — паттерн поведения объектов.

5.2 Назначение

Позволяет избежать привязки отправителя запроса к его получателю, предоставляя возможность обработать запрос нескольким объектам. Связывает объекты-получатели в цепочку и передает запрос по этой цепочке, пока он не будет обработан.

Структура



5.3 Участники

Handler — обработчик:

- определяет интерфейс для обработки запросов;
- (необязательно) реализует связь с преемником;

ConcreteHandler — конкретный обработчик:

- обрабатывает запрос, за который отвечает;
- имеет доступ к своему преемнику;
- если ConcreteHandler способен обработать запрос, то так и делает, если не может, то направляет его своему преемнику;

Client — клиент:

- отправляет запрос некоторому объекту ConcreteHandler в цепочке.

5.4 Отношения

Запрос, инициированный клиентом, продвигается по цепочке, пока некоторый объект ConcreteHandler не возьмет на себя ответственность за его обработку.

5.5 ЕО реализация

```
+package sandbox
+alias stdout org.eolang.io.stdout
+alias sprintf org.eolang.txt.sprintf
```

```
[nextHandler] > defaultHandler
[message] > process
  "" > @
```

```
[] > handler1
[message] > process
  if. > @
    message.eq "1"
    "one"
    ^.nextHandler.process message
defaultHandler > @
  handler2
```

```
[] > handler2
[message] > process
  if. > @
    message.eq "2"
    "two"
    ^.nextHandler.process message
defaultHandler > @
  handler3
```

```
[] > handler3
[message] > process
  if. > @
    message.eq "3"
    "three"
    ^.nextHandler.process message
defaultHandler > @
  handler4
```

```
[] > handler4
[message] > process
  if. > @
    message.eq "4"
    "four"
    ^.nextHandler.process message
defaultHandler > @
  defaultHandler
```

```
[args...] > appChain
  handler1 > hChain
  stdout > @
    sprintf
      "%s\n"
      hChain.process
      args.get 0
```


Входной параметр `args[0]` передается последовательно 4м обработчикам, каждый из которых обрабатывает свое значение (числа от 1 до 4 преобразуются в слова, если введен другой параметр возвращается пустая строка).

6. Команда(Command)

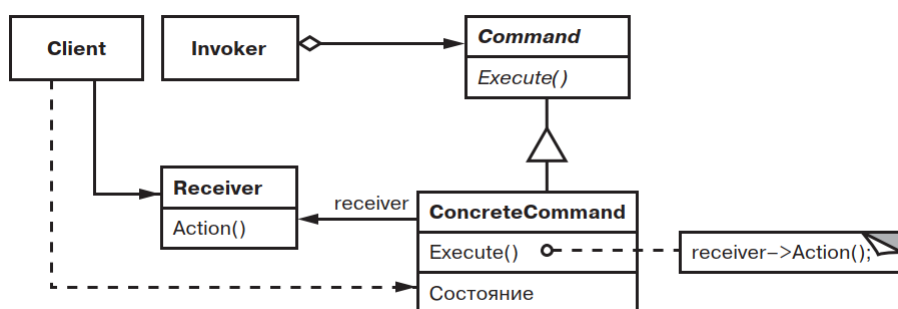
6.1 Название и классификация паттерна

Команда — паттерн поведения объектов.

6.2 Назначение

Инкапсулирует запрос в объекте, позволяя тем самым параметризовать клиенты для разных запросов, ставить запросы в очередь или протоколировать их, а также поддерживать отмену операций.

6.3 Структура



Участники

- Command — команда:
 - объявляет интерфейс для выполнения операции;
- ConcreteCommand — конкретная команда:
 - определяет связь между объектом-получателем Receiver и действием;
 - реализует операцию Execute путем вызова соответствующих операций объекта Receiver;
- Client — клиент:
 - создает объект класса ConcreteCommand и устанавливает его получателя;
- Invoker — инициатор:
 - обращается к команде для выполнения запроса;
- Receiver — получатель:
 - располагает информацией о способах выполнения операций, необходимых для удовлетворения запроса. В роли получателя может выступать любой класс.

6.4 Отношения

- клиент создает объект ConcreteCommand и устанавливает для него получателя;
- инициатор Invoker сохраняет объект ConcreteCommand;
- инициатор отправляет запрос, вызывая операцию команды Execute. Если поддерживается отмена выполненных действий, то ConcreteCommand перед вызовом Execute сохраняет информацию о состоянии, достаточную для выполнения отмены;
- объект ConcreteCommand вызывает операции получателя для выполнения запроса.

6.5 ЕО реализация

```
+package sandbox
+alias stdout org.eolang.io.stdout
+alias sprintf org.eolang.txt.sprintf
```

```
[receiver] > incCommand
[] > execute
add. > @
^.receiver
1
```

```
[receiver] > decCommand
[] > execute
sub. > @
^.receiver
1
```

```
[receiver] > doubleCommand
[] > execute
mul. > @
^.receiver
2
```

```
[cmd1 cmd2 cmd3] > invoker
```

```
[args...] > appCommandObjects
toInt. > receiver
args.get
0
```

```
invoker > inv
incCommand
receiver
decCommand
receiver
doubleCommand
receiver
```

```
stdout > @
sprintf
"%s - %s - %s"
inv.cmd1.execute
inv.cmd2.execute
inv.cmd3.execute
```

Объект `invoker` последовательно выполняет 3 команды над объектом `args[0]` (целое число):

- инкремент
- декремент
- удвоение

7. Наблюдатель(Observer)

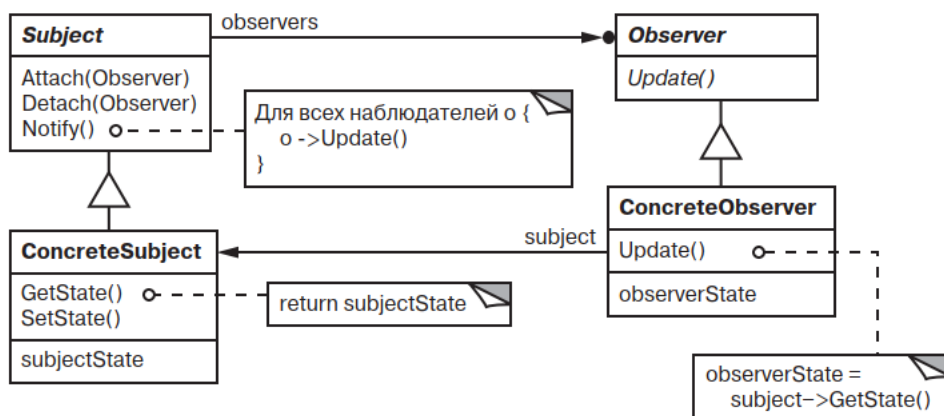
7.1 Название и классификация паттерна

Наблюдатель — паттерн поведения объектов.

7.2 Назначение

Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом и автоматически обновляются.

7.3 Структура



7.4 Участники

Subject — субъект:

- располагает информацией о своих наблюдателях. За субъектом может «следить» любое число наблюдателей;
- предоставляет интерфейс для присоединения и отделения наблюдателей;

Observer — наблюдатель:

- определяет интерфейс обновления для объектов, которые должны уведомляться об изменении субъекта;

ConcreteSubject — конкретный субъект:

- сохраняет состояние, представляющее интерес для конкретного наблюдателя ConcreteObserver;
- посылает информацию своим наблюдателям, когда происходит изменение;

ConcreteObserver — конкретный наблюдатель:

- хранит ссылку на объект класса ConcreteSubject;
- сохраняет данные, которые должны быть согласованы с данными субъекта;
- реализует интерфейс обновления, определенный в классе Observer, чтобы поддерживать согласованность с субъектом.

7.5 ЕО реализация

В ЕО все объекты имеют неизменяемое состояние. Исходя из назначения шаблона, его применение в ЕО бессмысленно.

8. Посредник(Mediator)

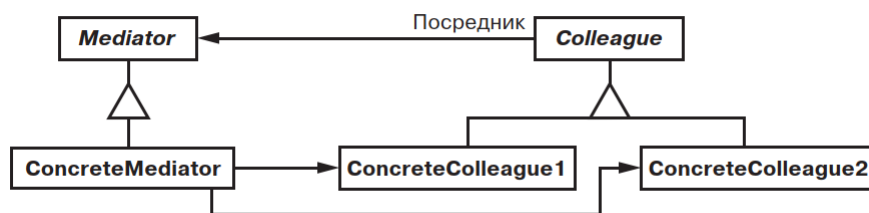
8.1 Название и классификация паттерна

Посредник — паттерн поведения объектов.

8.2 Назначение

Определяет объект, инкапсулирующий способ взаимодействия множества объектов. Посредник обеспечивает слабую связанность системы, избавляя объекты от необходимости явно ссылаться друг на друга и позволяя тем самым независимо изменять взаимодействия между ними.

8.3 Структура



8.4 Участники

Mediator — посредник:

- определяет интерфейс для обмена информацией с объектами Colleague;

ConcreteMediator — конкретный посредник:

- реализует кооперативное поведение, координируя действия объектов Colleague;
- владеет информацией о коллегах и подсчитывает их;

Классы Colleague — коллеги:

- каждый класс Colleague знает свой объект Mediator;
- все коллеги обмениваются информацией только с посредником во всех случаях, когда ему пришлось бы напрямую взаимодействовать с другими объектами.

8.5 Отношения

Коллеги посылают запросы посреднику и получают запросы от него. Посредник реализует кооперативное поведение путем переадресации каждого запроса подходящему коллеге (или нескольким коллегам).

8.6 ЕО реализация

Реализацией данного шаблона в ЕО может быть структура, в которой объект-медиатор либо имеет атрибут массив объектов-коллег, либо его атрибутами напрямую являются объекты-коллеги. Помимо это объект-медиатор содержит атрибуты, которые несут функцию обмена сообщениями между объектами коллегами:

```
[coll1 coll2 coll3] > mediator
[] > function1
[] > function2
[] > function3
```