



Lesson #2

Pattern “Strategy”

Паттерн «Strategy» («Стратегия»)

Название и классификация паттерна

Стратегия – паттерн поведения объектов. Другое название – «Policy» («Политика»)

Назначение

Определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми. Стратегия позволяет изменять алгоритмы независимо от клиентов, которые ими пользуются.

Мотивация

Предметная область может подразумевать наличие нескольких алгоритмов, выполняющих схожие действия. Например, алгоритмы разбиения на строки, сортировки, преобразования в заданный формат, конвертации и т.д. При этом объект остается неизменным, но в зависимости от контекста должен использовать тот или иной алгоритм.

Паттерн «Strategy» («Стратегия»)

Мотивация

При таком подходе возможны проблемы:

- класс клиента, который использует алгоритмы усложняется (все алгоритмы инкапсулируются в классе);
- необходимость поддержки разных алгоритмов в классе клиента, в т.ч. и устаревшие – усложняется добавление и модификация существующих алгоритмов.

Этих проблем можно избежать, если определить классы, инкапсулирующие различные алгоритмы внутри себя.

Инкапсулированный таким образом алгоритм называется *стратегией*.

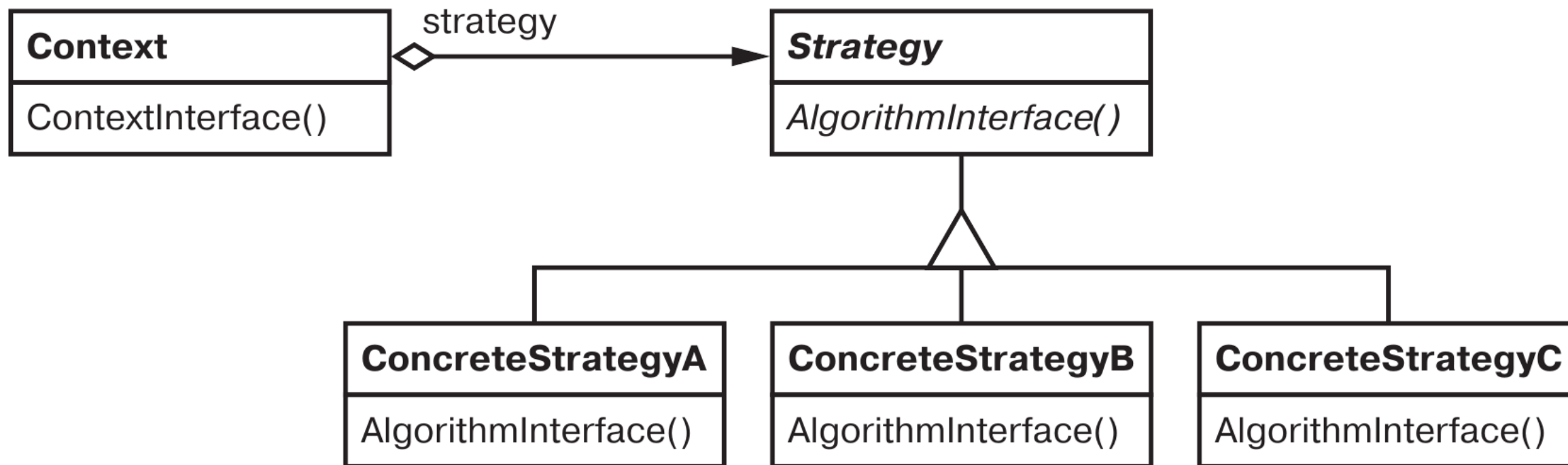
Паттерн «Strategy» («Стратегия»)

Применимость

- *наличие множества родственных классов, отличающихся только поведением.* Стратегия позволяет настроить класс одним из многих возможных вариантов поведения;
- *наличие нескольких разновидностей алгоритма.* Например, можно определить два варианта алгоритма, один из которых требует больше времени, а другой – больше памяти. Стратегии разрешается применять, когда варианты алгоритмов реализованы в виде иерархии классов;
- *в алгоритме содержатся данные, о которых клиент не должен «знать».* Используйте паттерн стратегия, чтобы не раскрывать сложные, специфичные для алгоритма структуры данных;
- *в классе определено много вариантов поведения, представленных разветвленными условными операторами.* В этом случае проще перенести код из ветвей в отдельные классы стратегий.

Паттерн «Strategy» («Стратегия»)

Структура



Паттерн «Strategy» («Стратегия»)

Участники

Strategy – стратегия

объявляет общий для всех поддерживаемых алгоритмов интерфейс (абстрактный класс). Класс `Context` пользуется этим интерфейсом для вызова конкретного алгоритма, определенного в классе `ConcreteStrategy`;

ConcreteStrategy – конкретная стратегия

реализует алгоритм, использующий интерфейс, объявленный в классе `Strategy`;

Context – контекст

настраивается объектом класса `ConcreteStrategy`;
хранит ссылку на объект класса `Strategy`;
может определять интерфейс, который позволяет объекту `Strategy` обращаться к данным контекста.

Паттерн «Strategy» («Стратегия»)

Отношения

- Классы *Strategy* и *Context* взаимодействуют для реализации выбранного алгоритма. Контекст может передать стратегии все необходимые алгоритму данные в момент его вызова. Вместо этого контекст может позволить обращаться к своим операциям в нужные моменты, передавая ссылку на самого себя операциям класса *Strategy*;
- Контекст переадресует запросы своих клиентов объекту-стратегии. Обычно клиент создает объект *ConcreteStrategy* и передает его контексту, после чего клиент взаимодействует исключительно с контекстом. Часто в распоряжении клиента находится несколько классов *ConcreteStrategy*, которые он может выбирать.

Паттерн «Strategy» («Стратегия»)

Результаты (основные достоинства паттерна)

- *семейства родственных алгоритмов*. Иерархия классов Strategy определяет семейство алгоритмов или вариантов поведения, которые можно повторно использовать в разных контекстах. Наследование позволяет вычленить общую для всех алгоритмов функциональность;
- *альтернатива порождению подклассов*. Наследование поддерживает многообразие алгоритмов или поведений. Можно напрямую породить от Context подклассы с различными поведением. Но при этом поведение жестко «зашивается» в класс Context. Реализации алгоритма и контекста смешиваются, что затрудняет понимание, сопровождение и расширение контекста. Кроме того, заменить алгоритм динамически уже не удастся. В результате вы получаете множество родственных классов, отличающихся только алгоритмом или поведением. Инкапсуляция алгоритма в отдельный класс Strategy позволяет изменять его независимо от контекста;

Паттерн «Strategy» («Стратегия»)

Результаты (основные достоинства паттерна)

- *стратегии позволяют избавиться от условных конструкций*. с паттерном стратегия удастся отказаться от условных операторов при выборе нужного поведения. Когда различные поведения помещаются в один класс, трудно выбрать нужное без применения условных операторов. Инкапсуляция же каждого поведения в отдельный класс Strategy решает эту проблему.
- *выбор реализации*. Стратегии могут предлагать различные реализации одного и того же поведения. Клиент вправе выбирать подходящую стратегию в зависимости от своих требований к быстродействию и памяти;

Паттерн «Strategy» («Стратегия»)

Результаты (основные недостатки паттерна)

- *клиенты должны знать о различных стратегиях.* Для выбора подходящей стратегии клиент должен понимать, чем отличаются разные стратегии. Поэтому наверняка придется раскрыть клиенту некоторые особенности реализации. Отсюда следует, что паттерн стратегия стоит применять лишь тогда, когда различия в поведении важны для клиента;
- *затраты на передачу информации между стратегией и контекстом.* Интерфейс Strategy совместно используется всеми подклассами ConcreteStrategy – какой бы сложной или тривиальной ни была их реализация. Поэтому вполне вероятно, что некоторые стратегии не будут пользоваться всей передаваемой им информацией, особенно простые;
- *увеличение числа объектов.* Применение стратегий увеличивает число объектов в приложении. Эти издержки можно сократить, если реализовать стратегии в виде объектов без состояния, которые могут совместно использоваться несколькими контекстами. Совместно используемые стратегии не должны сохранять состояние между вызовами.

Паттерн «Strategy» («Стратегия»)

Реализация

- *определение интерфейсов классов Strategy и Context.*
Интерфейсы классов Strategy и Context должны предоставить объекту класса ConcreteStrategy эффективный доступ к любым данным контекста, и наоборот.
- *объекты-стратегии можно не задавать.* Класс Context можно упростить, если для него нормально не иметь никакой стратегии. Прежде чем обращаться к объекту Strategy, объект Context проверяет наличие стратегии. Если да, то работа продолжается как обычно, в противном случае контекст реализует некое поведение по умолчанию. Преимущество такого подхода в том, что клиентам вообще не нужно иметь дело со стратегиями, если их устраивает поведение по умолчанию.

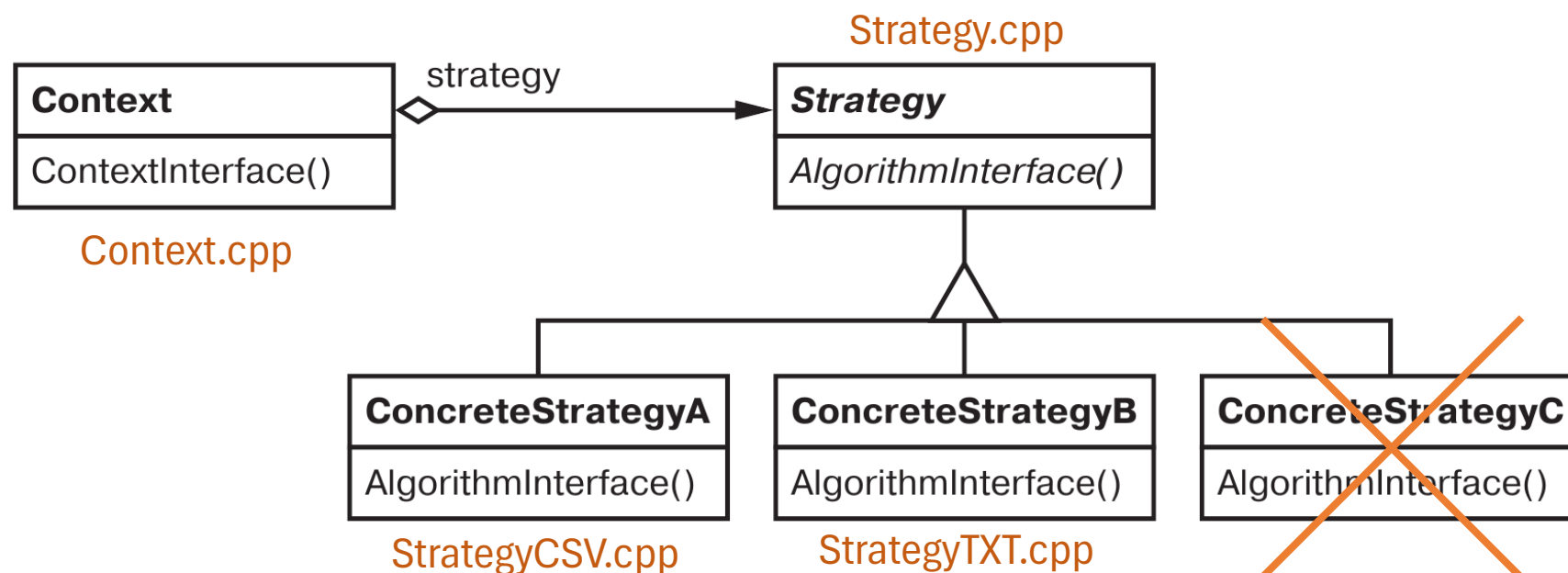
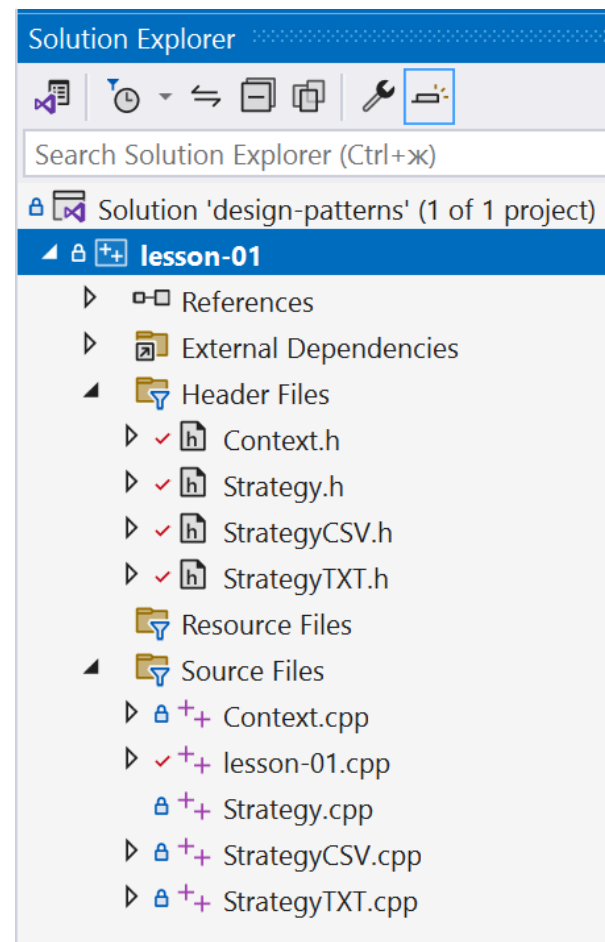
Родственные паттерны

- Приспособленец : объекты-стратегии в большинстве случаев подходят для применения паттерна приспособленец.

Паттерн «Strategy» («Стратегия»)

Пример кода

<https://github.com/VASoftLab/design-patterns.git>



Паттерн «Strategy» («Стратегия»)

Strategy.h

```
1      #pragma once
2      √ #include <string>
3      | #include <iostream>
4      | // Абстрактный класс стратегии
5      √ class Strategy
6      | {
7      |     public:
8      |         virtual ~Strategy() = default;
9      |         virtual std::string exportData() = 0;
10     | };

```

PATTERN "STRATEGY"

Паттерн «Strategy» («Стратегия»)

StrategyCSV.h

PATTERN "STRATEGY"

```
1  #pragma once
2  #include "Strategy.h"
3  // Класс стратегии для экспорта данных в CSV формате
4  class StrategyCSV : public Strategy
5  {
6  public:
7      StrategyCSV();
8      ~StrategyCSV() = default;
9  private:
10     std::string exportData() override;
11 }
```

Паттерн «Strategy» («Стратегия»)

StrategyTXT.h

PATTERN "STRATEGY"

```
1  #pragma once
2  #include "Strategy.h"
3  // Класс стратегии для экспорта данных в TXT формате
4  class StrategyTXT : public Strategy
5  {
6  public:
7      StrategyTXT();
8      ~StrategyTXT() = default;
9  private:
10     std::string exportData() override;
11 }
```

Паттерн «Strategy» («Стратегия»)

Context.h

```
1  #pragma once
2  #include <memory>
3  #include "Strategy.h"
4  // Класс контекста
5  class Context
6  {
7  private:
8      // Указатель на объект стратегии
9      std::unique_ptr<Strategy> _strategy;
10 public:
11     Context();
12     // Метод для задания стратегии
13     void setStrategy(std::unique_ptr<Strategy>&& strategy);
14     // Полиморфный метод для вывода данных
15     std::string exportData();
16 };
```


Паттерн «Strategy» («Стратегия»)

StrategyCSV.cpp

PATTERN "STRATEGY"

```
1  #include "StrategyCSV.h"
2
3  StrategyCSV::StrategyCSV( )
4  {
5      std::cout << "StrategyCSV constructor" << std::endl;
6  }
7  std::string StrategyCSV::exportData( )
8  {
9      std::cout << "Export data with CSV strategy" << std::endl;
10     return "CSV";
11 }
```

Паттерн «Strategy» («Стратегия»)

StrategyTXT.cpp

PATTERN «STRATEGY»

```
1  #include "StrategyTXT.h"
2  [
3  StrategyTXT::StrategyTXT( )
4  {
5      std::cout << "StrategyTXT constructor" << std::endl;
6  }
7  std::string StrategyTXT::exportData( )
8  {
9      std::cout << "Export data with TXT strategy" << std::endl;
10     return "TXT";
11 }
```

Паттерн «Strategy» («Стратегия»)

Context.cpp

```
1  #include "Context.h"
2  Context::Context()
3  {
4      _strategy = NULL;
5  }
6  void Context::setStrategy(std::unique_ptr<Strategy>&& strategy)
7  {
8      if (strategy)
9          _strategy = std::move(strategy);
10 }
11 std::string Context::exportData()
12 {
13     if (_strategy)
14         return _strategy->exportData();
15     else
16         return "ERROR";
17 }
18
```

Паттерн «Strategy» («Стратегия»)

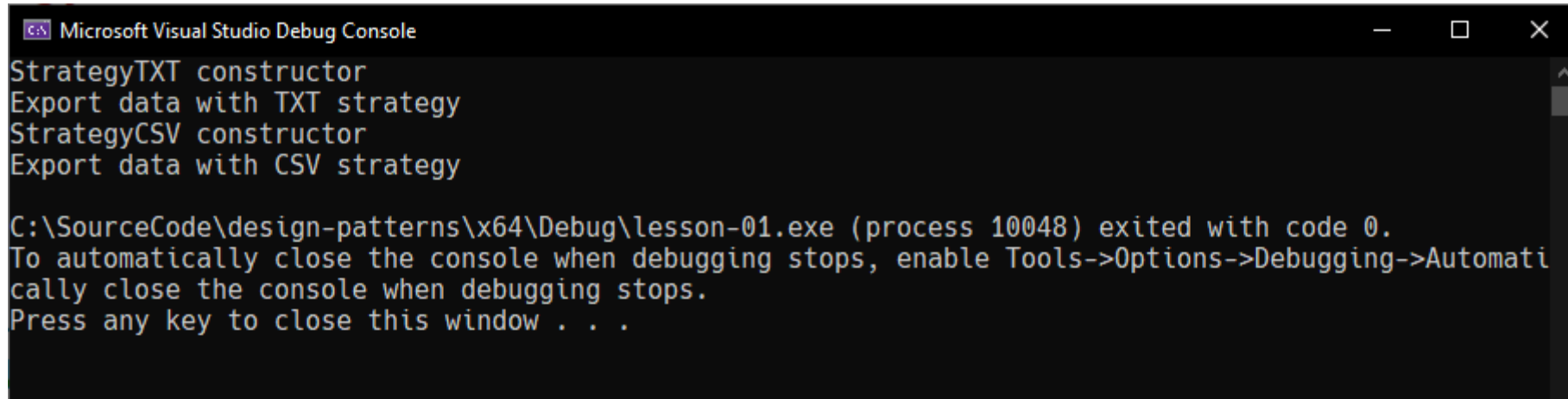
lesson-01.cpp

PATTERN "STRATEGY"

```
1  ✓ #include <iostream>
2    #include "Context.h"
3    #include "StrategyCSV.h"
4    #include "StrategyTXT.h"
5
6  ✓ int main()
7    {
8      // Создание объекта контекста
9      std::unique_ptr<Context> context = std::make_unique<Context>( );
10     // Назначение стратегии A (TXT)
11     context->setStrategy(std::make_unique<StrategyTXT>( ));
12     context->exportData( );
13     // Назначение стратегии B (CSV)
14     context->setStrategy(std::make_unique<StrategyCSV>( ));
15     context->exportData( );
16 }
```

Паттерн «Strategy» («Стратегия»)

Результаты работы



```
Microsoft Visual Studio Debug Console
StrategyTXT constructor
Export data with TXT strategy
StrategyCSV constructor
Export data with CSV strategy

C:\SourceCode\design-patterns\x64\Debug\lesson-01.exe (process 10048) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automati
cally close the console when debugging stops.
Press any key to close this window . . .
```

to be continued...