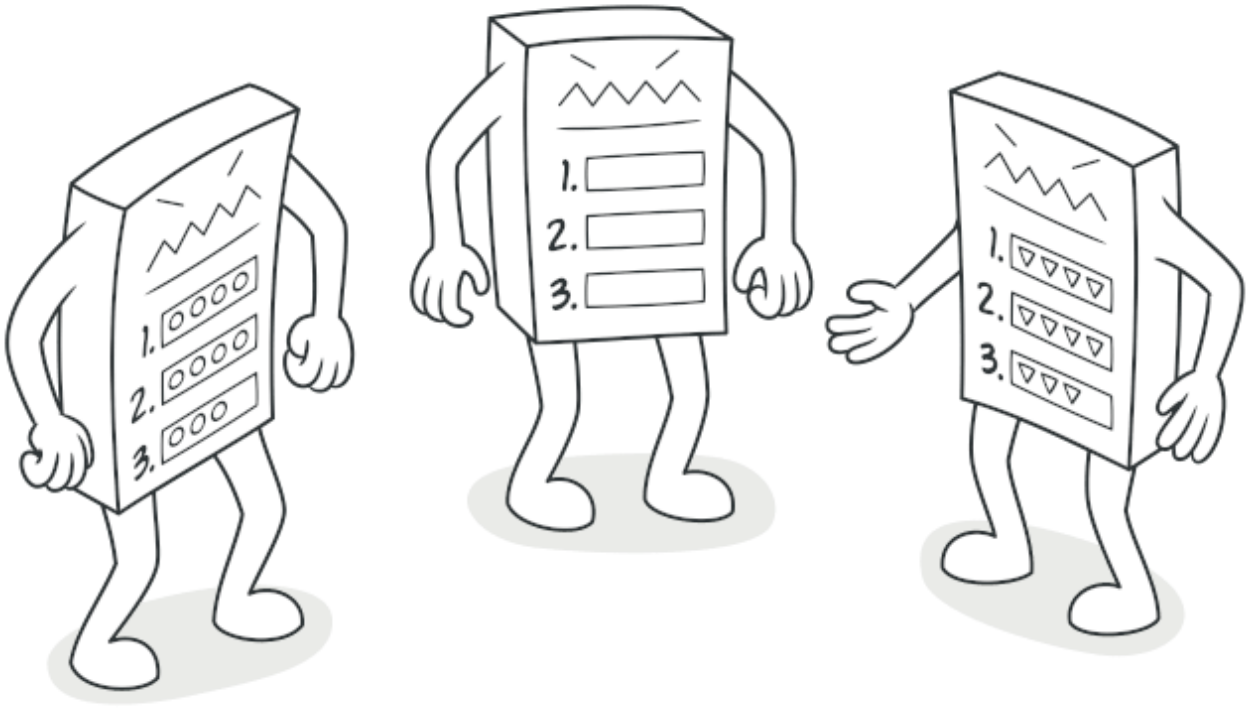
Шаблонный метод

Также известен как: Template Method

Суть паттерна

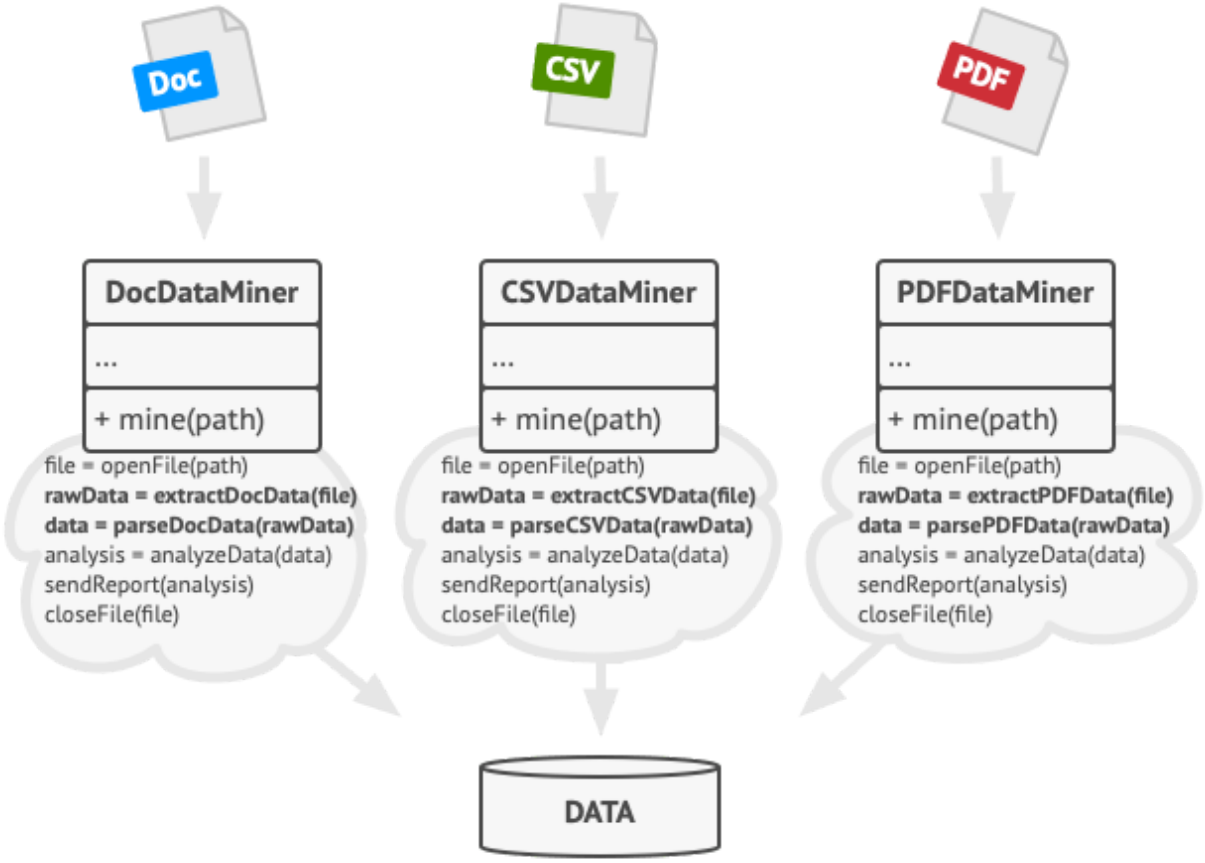
Шаблонный метод — это поведенческий паттерн проектирования, который определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Паттерн позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.



Проблема

Вы пишете программу для дата-майнинга в офисных документах. Пользователи будут загружать в неё документы в разных форматах (PDF, DOC, CSV), а программа должна извлекать из них полезную информацию.

В первой версии вы ограничились только обработкой DOC-файлов. В следующей версии добавили поддержку CSV. А через месяц прикрутили работу с PDF-документами.



Классы дата-майнинга содержат много дублирования.

В какой-то момент вы заметили, что код всех трёх классов обработки документов хоть и отличается в части работы с файлами, но содержат довольно много общего в части самого извлечения данных. Было бы здорово избавится от повторной реализации алгоритма извлечения данных в каждом из классов.

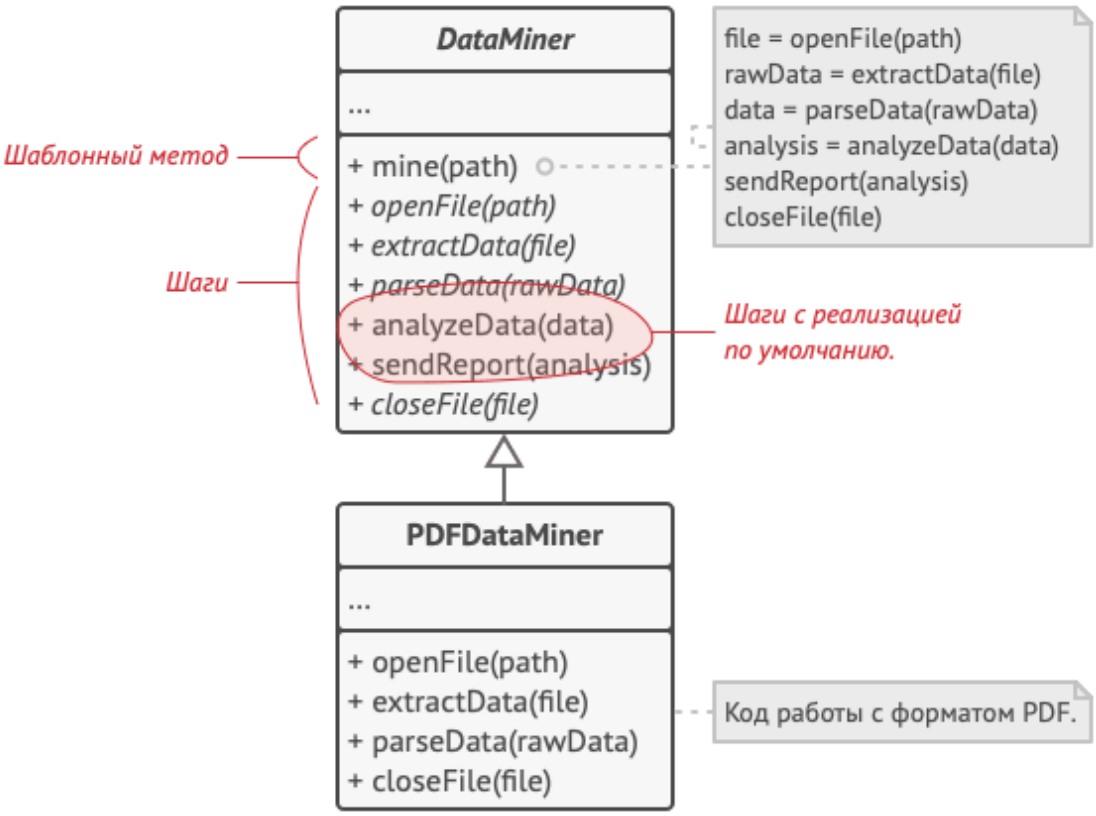
К тому же остальной код, работающий с объектами этих классов, наполнен условиями, проверяющими тип обработчика перед началом работы. Весь этот код можно упростить, если слить все три класса воедино либо свести их к общему интерфейсу.

Решение

Паттерн Шаблонный метод предлагает разбить алгоритм на последовательность шагов, описать эти шаги в отдельных методах и вызывать их в одном *шаблонном* методе друг за другом.

Это позволит подклассам переопределять некоторые шаги алгоритма, оставляя без изменений его структуру и остальные шаги, которые для этого подкласса не так важны.

В нашем примере с дата-майнингом мы можем создать общий базовый класс для всех трёх алгоритмов. Этот класс будет состоять из шаблонного метода, который последовательно вызывает шаги разбора документов.



Шаблонный метод разбивает алгоритм на шаги, позволяя подклассам переопределить некоторые из них.

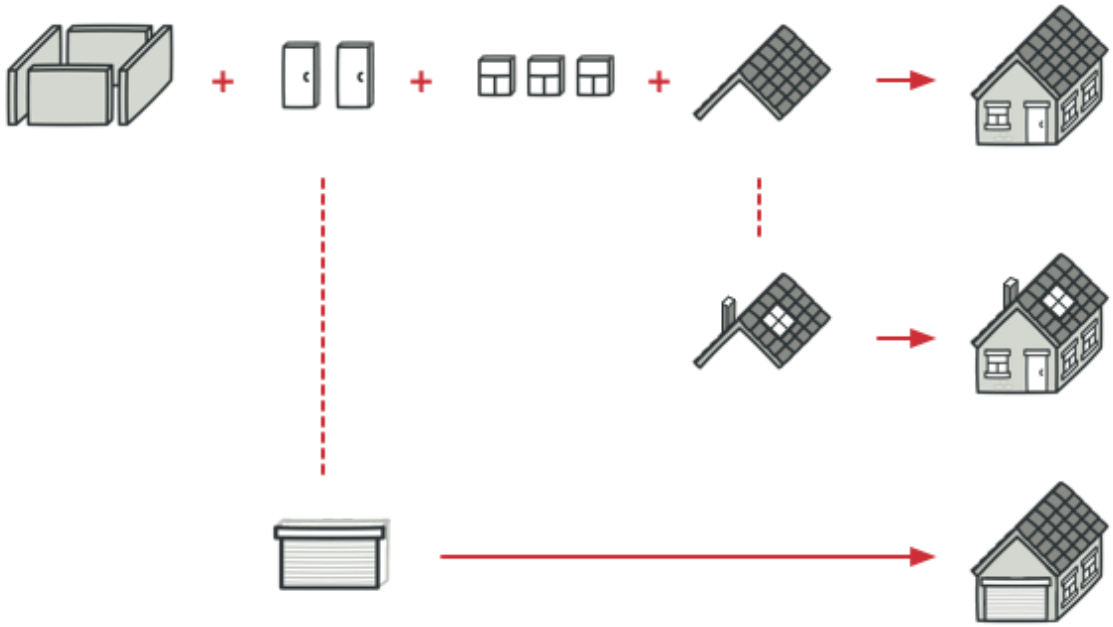
Для начала шаги шаблонного метода можно сделать абстрактными. Из-за этого все подклассы должны будут реализовать каждый из шагов по-своему. В нашем случае все подклассы и так содержат реализацию каждого из шагов, поэтому ничего дополнительно делать не нужно.

По-настоящему важным является следующий этап. Теперь мы можем определить общее для всех классов поведение и вынести его в суперкласс. В нашем примере шаги открытия, считывания и закрытия могут отличаться для разных типов документов, поэтому останутся абстрактными. А вот одинаковый для всех типов документов код обработки данных переедет в базовый класс.

Как видите, у нас получилось два вида шагов: *абстрактные*, которые каждый подкласс обязательно должен реализовать, а также шаги *с реализацией по умолчанию*, которые можно переопределять в подклассах, но не обязательно.

Но есть и третий тип шагов — *хуки*: их не обязательно переопределять, но они не содержат никакого кода, выглядя как обычные методы. Шаблонный метод останется рабочим, даже если ни один подкласс не переопределит такой хук. Однако, хук даёт подклассам дополнительные точки «вклинивания» в шаблонный метод.

Аналогия из жизни

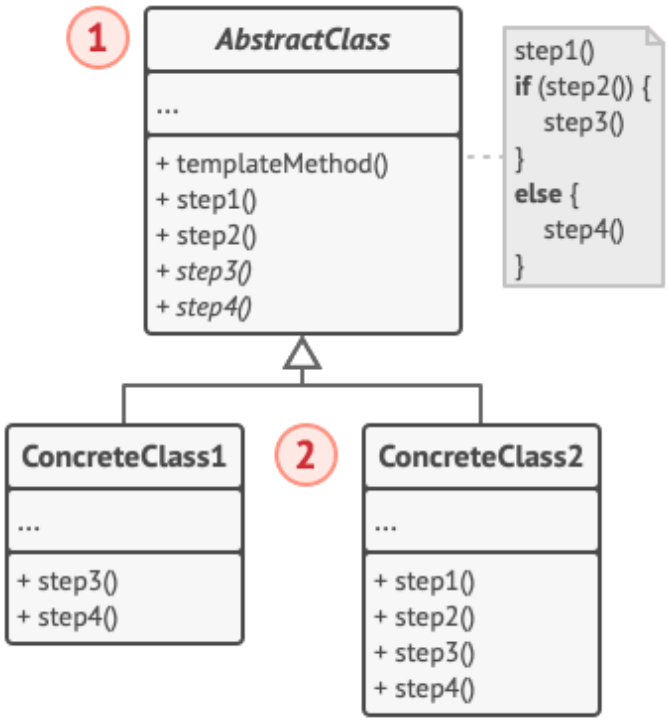


Проект типового дома могут немного изменить по желанию клиента.

Строители используют подход, похожий на шаблонный метод при строительстве типовых домов. У них есть основной архитектурный проект, в котором расписаны шаги строительства: заливка фундамента, постройка стен, перекрытие крыши, установка окон и так далее.

Но, несмотря на стандартизацию каждого этапа, строители могут вносить небольшие изменения на любом из этапов, чтобы сделать дом чуточку непохожим на другие.

Структура

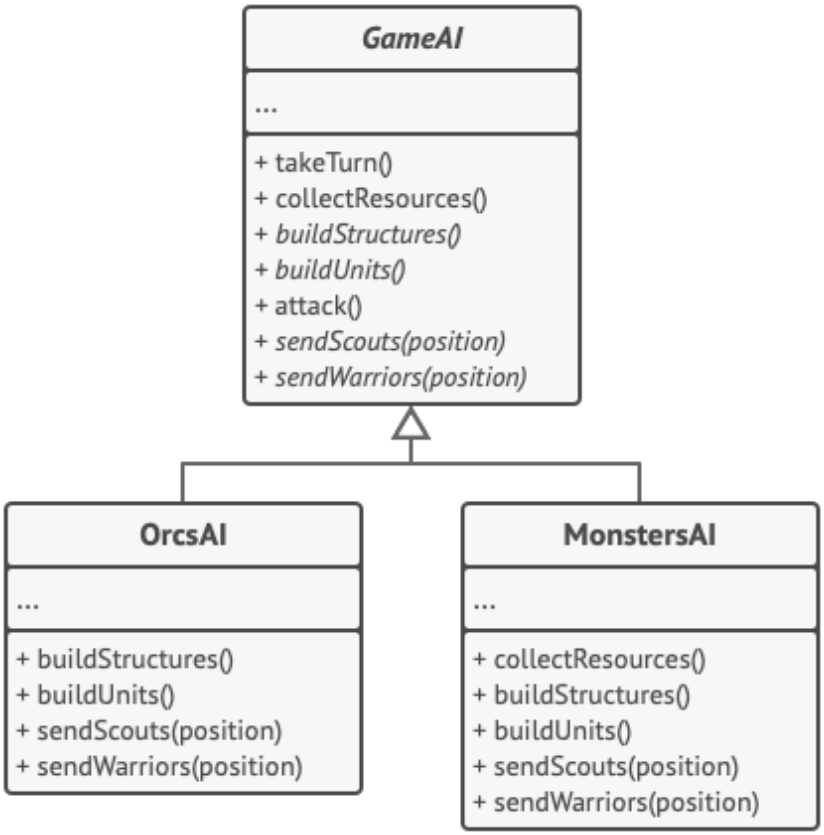


1. **Абстрактный класс** определяет шаги алгоритма и содержит шаблонный метод, состоящий из вызовов этих шагов. Шаги могут быть как абстрактными, так и содержать реализацию по умолчанию.

2. **Конкретный класс** переопределяет некоторые (или все) шаги алгоритма. Конкретные классы не переопределяют сам шаблонный метод.

Псевдокод

В этом примере **Шаблонный метод** используется как заготовка для стандартного искусственного интеллекта в простой игре-стратегии. Для введения в игру новой расы достаточно создать подкласс и реализовать в нём недостающие методы.



Пример классов искусственного интеллекта для простой игры.

Все расы игры будут содержать примерно такие же типы юнитов и строений, поэтому структура ИИ будет одинаковой. Но разные расы могут по-разному реализовать эти шаги. Так, например, орки будут агрессивней в атаке, люди — более активны в защите, а дикие монстры вообще не будут заниматься строительством.

```
class GameAI is
    // Шаблонный метод должен быть задан в базовом классе. Он
    // состоит из вызовов методов в определённом порядке. Чаще
    // всего эти методы являются шагами некоего алгоритма.
    method turn() is
        collectResources()
```

```
        buildStructures()
        buildUnits()
        attack()

    // Некоторые из этих методов могут быть реализованы прямо в
    // базовом классе.
    method collectResources() is
        foreach (s in this.builtStructures) do
            s.collect()

    // А некоторые могут быть полностью абстрактными.
    abstract method buildStructures()
    abstract method buildUnits()

    // Кстати, шаблонных методов в классе может быть несколько.
    method attack() is
        enemy = closestEnemy()
        if (enemy == null)
            sendScouts(map.center)
        else
            sendWarriors(enemy.position)

    abstract method sendScouts(position)
    abstract method sendWarriors(position)

// Подклассы могут предоставлять свою реализацию шагов
// алгоритма, не изменяя сам шаблонный метод.
class OrcsAI extends GameAI is
    method buildStructures() is
        if (there are some resources) then
            // Строить фермы, затем бараки, а потом цитадель.

    method buildUnits() is
        if (there are plenty of resources) then
            if (there are no scouts)
                // Построить раба и добавить в группу
                // разведчиков.
            else
                // Построить пехотинца и добавить в группу
                // воинов.

    // ...

    method sendScouts(position) is
        if (scouts.length > 0) then
            // Отправить разведчиков на позицию.

    method sendWarriors(position) is
        if (warriors.length > 5) then
            // Отправить воинов на позицию.

// Подклассы могут не только реализовывать абстрактные шаги, но
```

```
// и переопределять шаги, уже реализованные в базовом классе.
class MonstersAI extends GameAI is
    method collectResources() is
        // Ничего не делать.

    method buildStructures() is
        // Ничего не делать.

    method buildUnits() is
        // Ничего не делать.
```

Применимость

Когда подклассы должны расширять базовый алгоритм, не меняя его структуры.

Шаблонный метод позволяет подклассам расширять определённые шаги алгоритма через наследование, не меняя при этом структуру алгоритмов, объявленную в базовом классе.

Когда у вас есть несколько классов, делающих одно и то же с незначительными отличиями. Если вы редактируете один класс, то приходится вносить такие же правки и в остальные классы.

Паттерн шаблонный метод предлагает создать для похожих классов общий суперкласс и оформить в нём главный алгоритм в виде шагов. Отличающиеся шаги можно переопределить в подклассах.

Это позволит убрать дублирование кода в нескольких классах с похожим поведением, но отличающихся в деталях.

Шаги реализации

1. Изучите алгоритм и подумайте, можно ли его разбить на шаги. Прикиньте, какие шаги будут стандартными для всех вариаций алгоритма, а какие — изменяющимися.

2. Создайте абстрактный базовый класс. Определите в нём шаблонный метод. Этот метод должен состоять из вызовов шагов алгоритма. Имеет смысл сделать шаблонный метод финальным, чтобы подклассы не могли переопределить его (если ваш язык программирования это позволяет).

3. Добавьте в абстрактный класс методы для каждого из шагов алгоритма. Вы можете сделать эти методы абстрактными или добавить какую-то реализацию по умолчанию. В первом случае все подклассы *должны* будут реализовать эти методы, а во втором — только если реализация шага в подклассе отличается от стандартной версии.

4. Подумайте о введении в алгоритм хуков. Чаще всего, хуки располагают между основными шагами алгоритма, а также до и после всех шагов.

5. Создайте конкретные классы, унаследовав их от абстрактного класса. Реализуйте в них все недостающие шаги и хуки.

Преимущества и недостатки

Облегчает повторное использование кода.

Вы жёстко ограничены скелетом существующего алгоритма.

Вы можете нарушить *принцип подстановки Барбары Лисков*, изменяя базовое поведение одного из шагов алгоритма через подкласс.

С ростом количества шагов шаблонный метод становится слишком сложно поддерживать.

Отношения с другими паттернами

- **Фабричный метод** можно рассматривать как частный случай **Шаблонного метода**. Кроме того, *Фабричный метод* нередко бывает частью большого класса с *Шаблонными методами*.

- **Шаблонный метод** использует наследование, чтобы расширять части алгоритма. **Стратегия** использует делегирование, чтобы изменять выполняемые алгоритмы на лету. *Шаблонный метод* работает на уровне классов. *Стратегия* позволяет менять логику отдельных объектов.