Реализация паттерна State

Выполнил: Студент группы 13ПИ

Бакаев Никита

Содержание

1) Название и классификация………………………………………………………………………….3

2) Назначение…………………………………………………………………………………………………..3

3) Структура(UML диаграмма классов)……………………………………………………………3

4) Описание классов-участников и их отношений …………………………………………4

5) Способы реализации паттерна …………………………………………………………………..4

6) Описание примера реализации ………………………………………………………………….5

7) UML диаграмма классов …………………………………………………………………………….11

8) UML диаграмма взаимодействий……………………………………………………………….12

9) Плюсы применения паттерна …………………………………………………………………….13

10) Минусы применения паттерна …………………………………………………………………13

# Название и классификация

**Состояние** (англ. *State*) — поведенческий шаблон проектирования

# Назначение

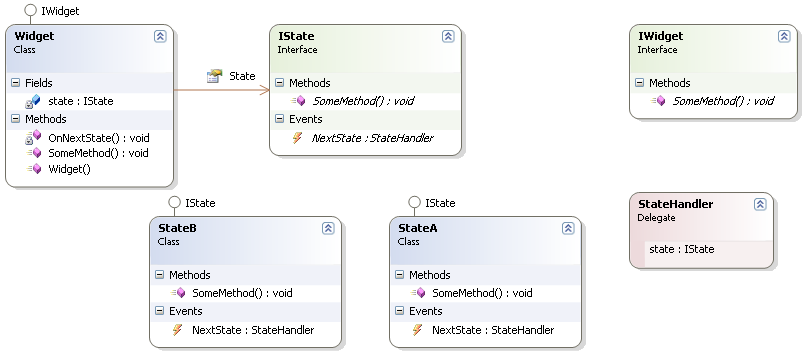
Используется в тех случаях, когда во время выполнения программы объект должен менять свое поведение в зависимости от своего состояния.

## Решаемая проблема

Поведение объекта зависит от его состояния и должно изменяться во время выполнения программы. Такую схему можно реализовать, применив множество условных операторов: на основе анализа текущего состояния объекта предпринимаются определенные действия. Однако при большом числе состояний условные операторы будут разбросаны по всему коду, и такую программу будет трудно поддерживать.

# Структура

## UML диаграмма классов и взаимодействий



Паттерн состоит из 3 блоков:

* Widget – класс, объекты которого должны менять свое поведение в зависимости от состояния.
* IState – интерфейс, который должен реализовать каждое из конкретных состояний. Через этот интерфейс объект Widget взаимодействует с состоянием, делегируя ему вызовы методов. Интерфейс должен содержать средства для обратной связи с объектом, поведение которого нужно изменить. Для этого используется событие (паттерн Publisher - Subscriber). Это необходимо для того, чтобы в процессе выполнения программы заменять объект состояния при появлении событий. Возможны случаи, когда сам Widget периодически опрашивает объект состояние на наличие перехода.
* StateA … StateZ – классы конкретных состояний. Должны содержать информацию о том, при каких условиях и в какие состояния может переходить объект из текущего состояния. Например, из StateA объект может переходить в состояние StateB и StateC, а из StateB – обратно в StateA и так далее. Объект одного из них должен содержать Widget при создании.

**Способы реализации паттерна**

Паттерн State решает указанную проблему следующим образом:

* Вводит класс Context, в котором определяется интерфейс для внешнего мира.
* Вводит абстрактный класс State.
* Представляет различные "состояния" конечного автомата в виде подклассов State.
* В классе Context имеется указатель на текущее состояние, который изменяется при изменении состояния конечного автомата.

Паттерн State не определяет, где именно определяется условие перехода в новое состояние. Существует два варианта: класс Context или подклассы State. Преимущество последнего варианта заключается в простоте добавления новых производных классов. Недостаток заключается в том, что каждый подкласс State для осуществления перехода в новое состояние должен знать о своих соседях, что вводит зависимости между подклассами.

## **Структура паттерна State**

Класс Context определяет внешний интерфейс для клиентов и хранит внутри себя ссылку на текущее состояние объекта State. Интерфейс абстрактного базового класса State повторяет интерфейс Context за исключением одного дополнительного параметра - указателя на экземплярContext. Производные от State классы определяют поведение, специфичное для конкретного состояния. Класс "обертка" Context делегирует все полученные запросы объекту "текущее состояние", который может использовать полученный дополнительный параметр для доступа к экземпляру Context.

## **Использование паттерна State**

* Определите существующий или создайте новый класс-"обертку" Context, который будет использоваться клиентом в качестве "конечного автомата".
* Создайте базовый класс State, который повторяет интерфейс класса Context. Каждый метод принимает один дополнительный параметр: экземпляр классаContext. Класс State может определять любое полезное поведение "по умолчанию".
* Создайте производные от State классы для всех возможных состояний.
* Класс-"обертка" Context имеет ссылку на объект "текущее состояние".
* Все полученные от клиента запросы класс Context просто делегирует объекту "текущее состояние", при этом в качестве дополнительного параметра передается адрес объекта Context.
* Используя этот адрес, в случае необходимости методы класса State могут изменить "текущее состояние" класса Context.

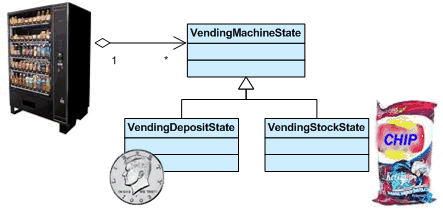
## **Особенности паттерна State**

* Объекты класса State часто бывают одиночками.
* Flyweight показывает, как и когда можно разделять объекты State.
* Паттерн Interpreter может использовать State для определения контекстов при синтаксическом разборе.
* Паттерны State и Bridge имеют схожие структуры за исключением того, что Bridge допускает иерархию классов-конвертов (аналогов классов-"оберток"), а State-нет. Эти паттерны имеют схожие структуры, но решают разные задачи: State позволяет объекту изменять свое поведение в зависимости от внутреннего состояния, в то время как Bridge разделяет абстракцию от ее реализации так, что их можно изменять независимо друг от друга.
* Реализация паттерна State основана на паттерне Strategy. Различия заключаются в их назначении.

## **Пример паттерна State**

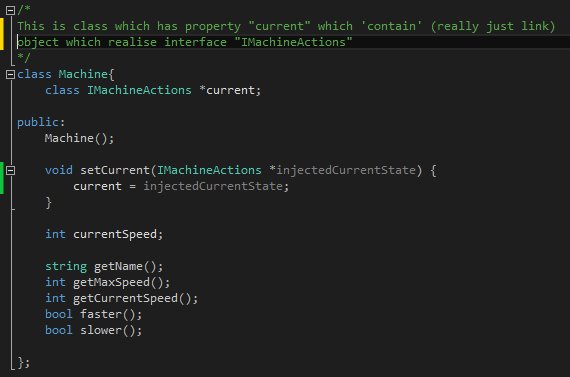
Паттерн State позволяет объекту изменять свое поведение в зависимости от внутреннего состояния. Похожая картина может наблюдаться в работе торгового автомата. Автоматы могут иметь различные состояния в зависимости от наличия товаров, суммы полученных монет, возможности размена денег и т.д. После того как покупатель выбрал и оплатил товар, возможны следующие ситуации (состояния):

* Выдать покупателю товар, выдавать сдачу не требуется.
* Выдать покупателю товар и сдачу.
* Покупатель товар не получит из-за отсутствия достаточной суммы денег.
* Покупатель товар не получит из-за его отсутствия.

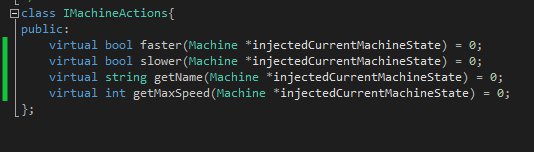


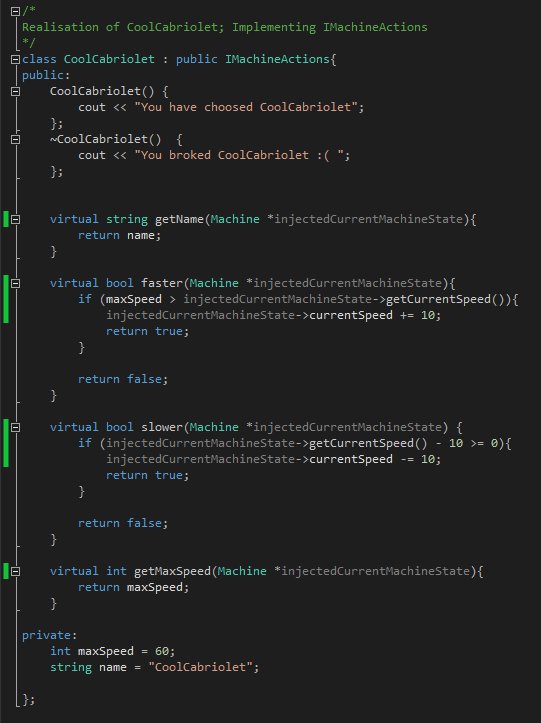
Пример реализации паттерна:

Основной класс (машина), содержащий состояние (current)

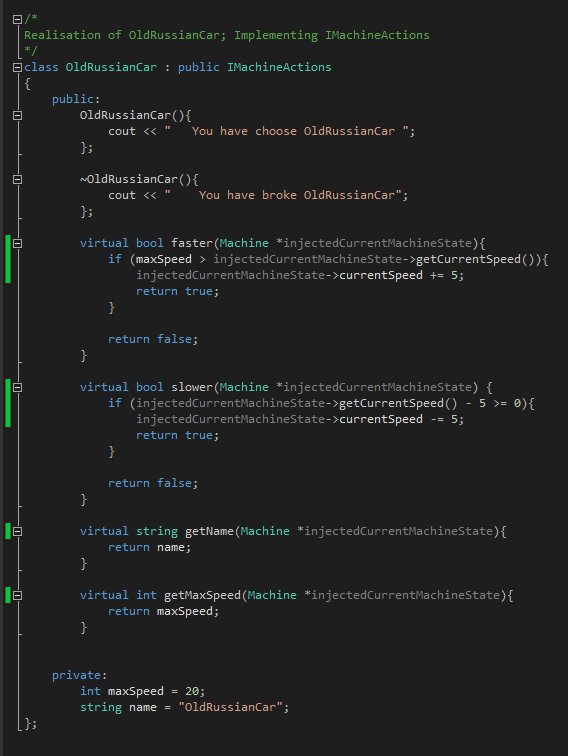


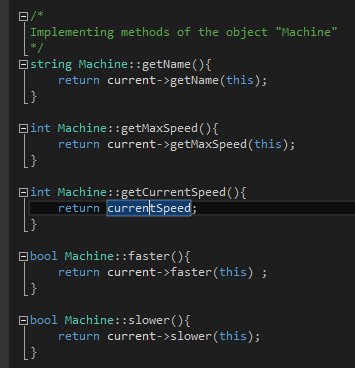
Интерфейс, который должна реализовать каждая конкретная имлементация машины



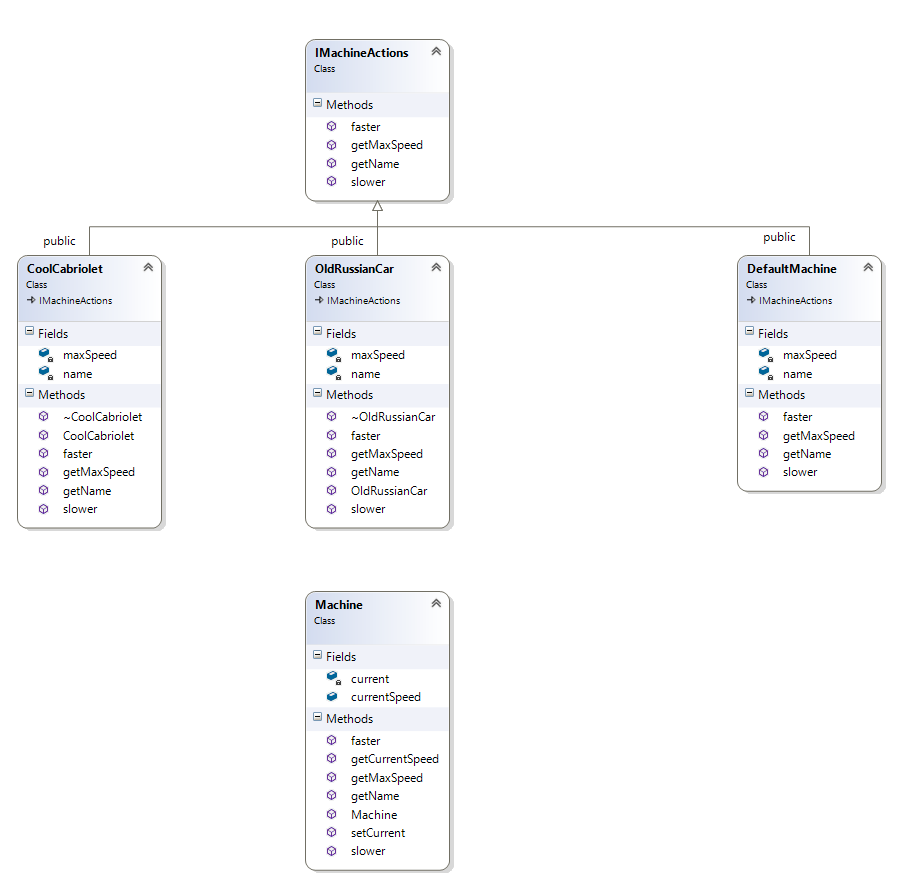
Первый пример имплементации машины

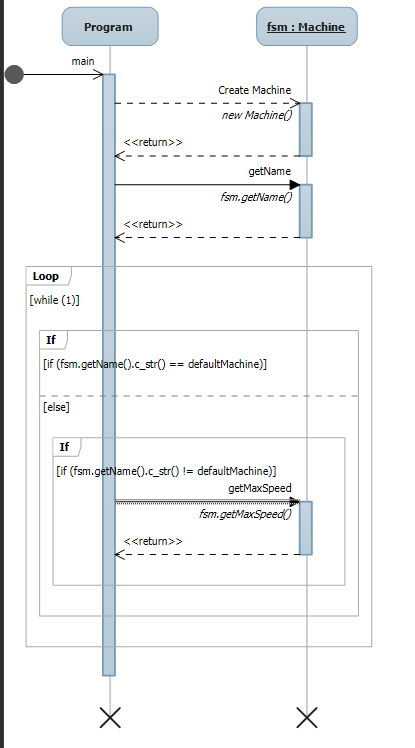
Второй пример имплементации машины





В ходе работы программы, мы вызываем у текущего состояния методы, которые он обязан был реализовать в ходе имплементации IMachineActions

Таким образом, мы можем менять конкретную реализации и поведения нашего класса “машина” в зависимости от конкретно в текущий момент выбранной машины. Таким образом, прозрачно для внешнего пользователя делегируется бизнес-логика на конкретную реализацию, а снаружи взаимодействие происходит через заранее подготовленный interface. 

****

**Плюсы**:

* Смена поведения объекта в Runtime в зависимости от каких-либо факторов
* Программировать на уровне интерфейсов хорошо

**Минусы**:

* В некоторых случаях, когда мы хотим перейти между разными состояниями во реализации состояния, необходимо явно указывать необходимую реализацию