Stateless:

управляем состоянием объектов

Обо мне

Владислав Шакиров

Разработчик

Разработчик в Райффайзенбанк. Разрабатываю на .Net уже около 5 лет. Занимаюсь разработкой Backend и Desktop-приложений, а так же стараюсь делать архитектуру приложений проще и понятнее ☺

План доклада

Райффайзен

- Что же такое
  конечный автомат?
- Обзор библиотеки Stateless
- Пример реализации и билдер автомата

1 Что такое конечный автомат

Коне́чный автома́т — абстрактный автомат, число возможных внутренних состояний которого конечно.

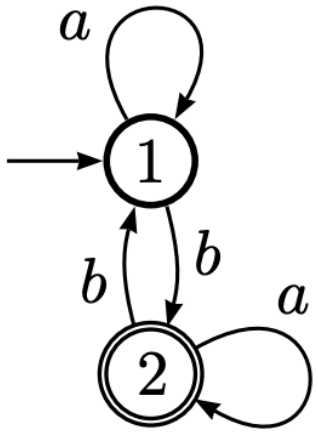
Диаграмма состояний (или иногда **граф переходов**) — графическое представление множества состояний и функции переходов. Представляет собой размеченный ориентированный граф, вершины которого — состояния КА, дуги — переходы из одного состояния в другое.

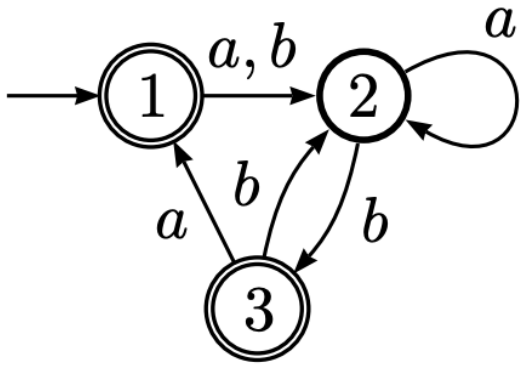
Таблица переходов — табличное представление функции. Обычно в такой таблице каждой строке соответствует одно состояние, а столбцу — один допустимый входной символ. В ячейке на пересечении строки и столбца записывается состояние, в которое должен перейти автомат, если в данном состоянии он считал данный входной символ.

Конечный автомат

1,2,3 – состояния конечного автомата

a, b – возможные переходов





Для чего используется

- Описание процесса
- Модель вычислений
- Модели поведения
- Реализация ботов

У нас есть лифт…

Райффайзен

Stateless лифт

```
public class StatelessLift
{
    4 references | 0 exceptions
    public int Floor { get; set; }
    1 reference | 0 exceptions
    public bool Active { get; set; }
    1 reference | 0 exceptions
    public bool DoorIsOpen { get; set; }
}
```

У нас есть лифт…

Райффайзен

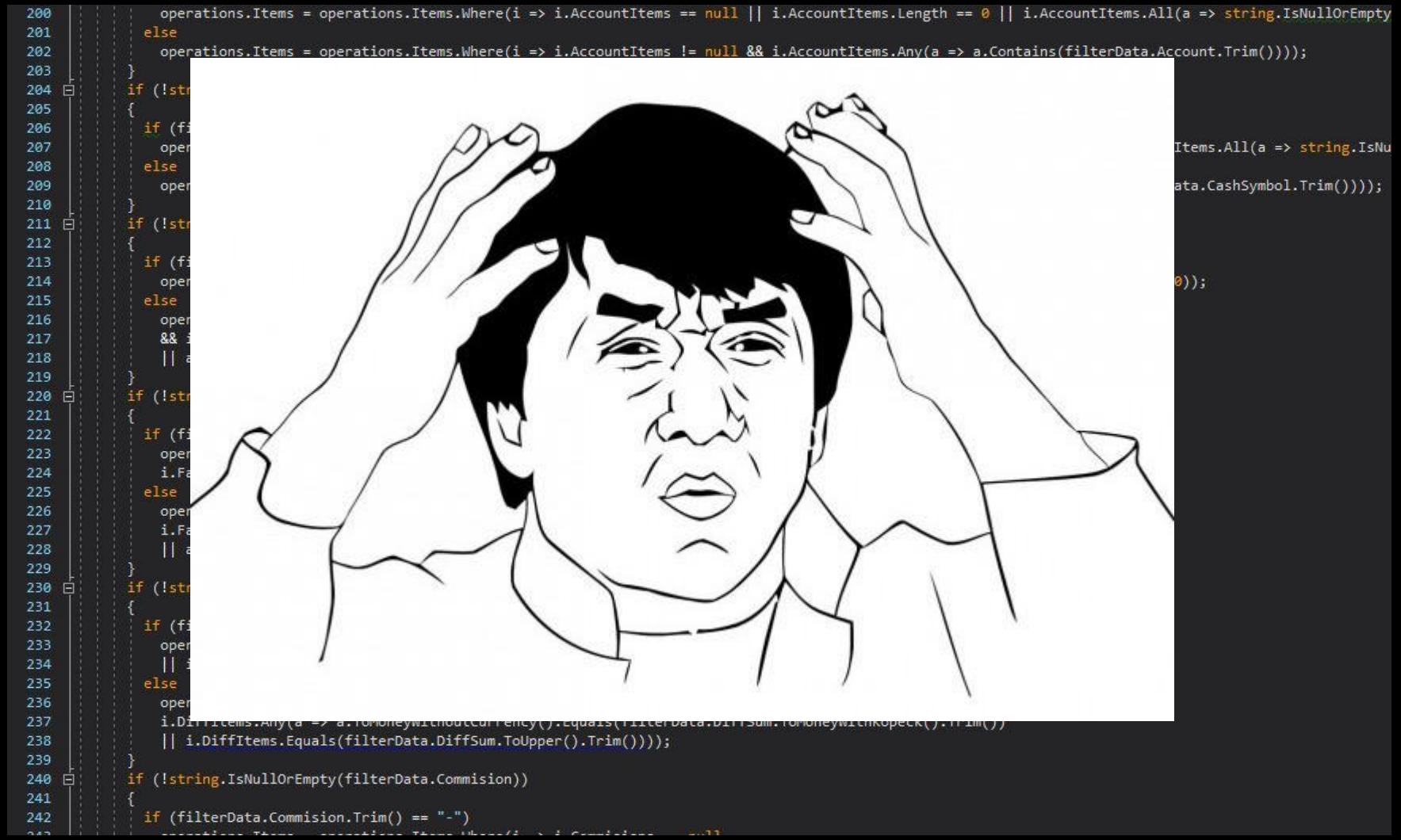
Stateless лифт

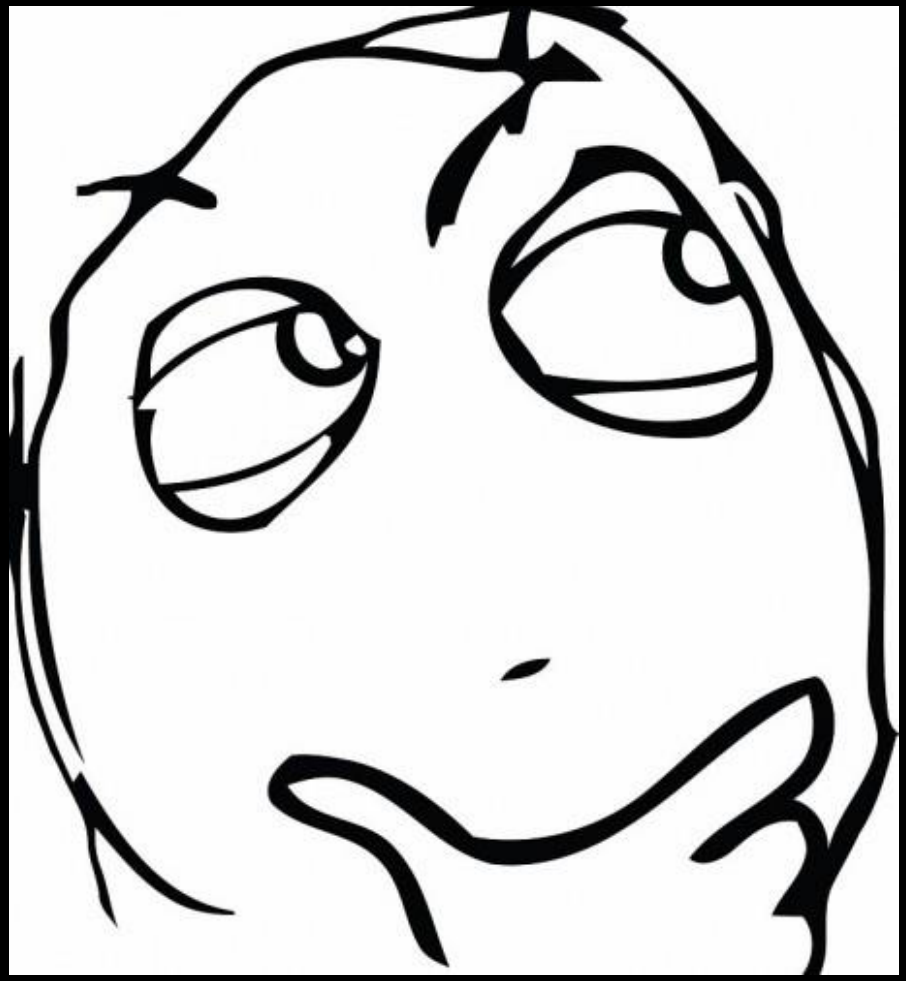
```
public void GoToFloor(int floor)
{
  if (Floor == 1)
  {
    switch (floor)
    {
      case 2:
        Console.WriteLine("Go to 2 floor");
        DoorIsOpen = false;
        Floor = 2;
        DoorIsOpen = true;
        break;
      case 3:
        Console.WriteLine("Go to 3 floor");
        DoorIsOpen = false;
        Floor = 3;
        DoorIsOpen = true;
        break;
    }
  }
}
```

```
if (!Active)
{
  throw new Exception("Lift is moving!");
}

if (!DoorIsOpen)
{
  DoorIsOpen = true;
}
```

У нас есть лифт…

Stateless лифт





Состояние

Состояние – это совокупность изменяемых параметров объекта

Триггеры

Триггер – это событие, по которому происходит переход из одного состояния в другое

У нас есть лифт 2.0

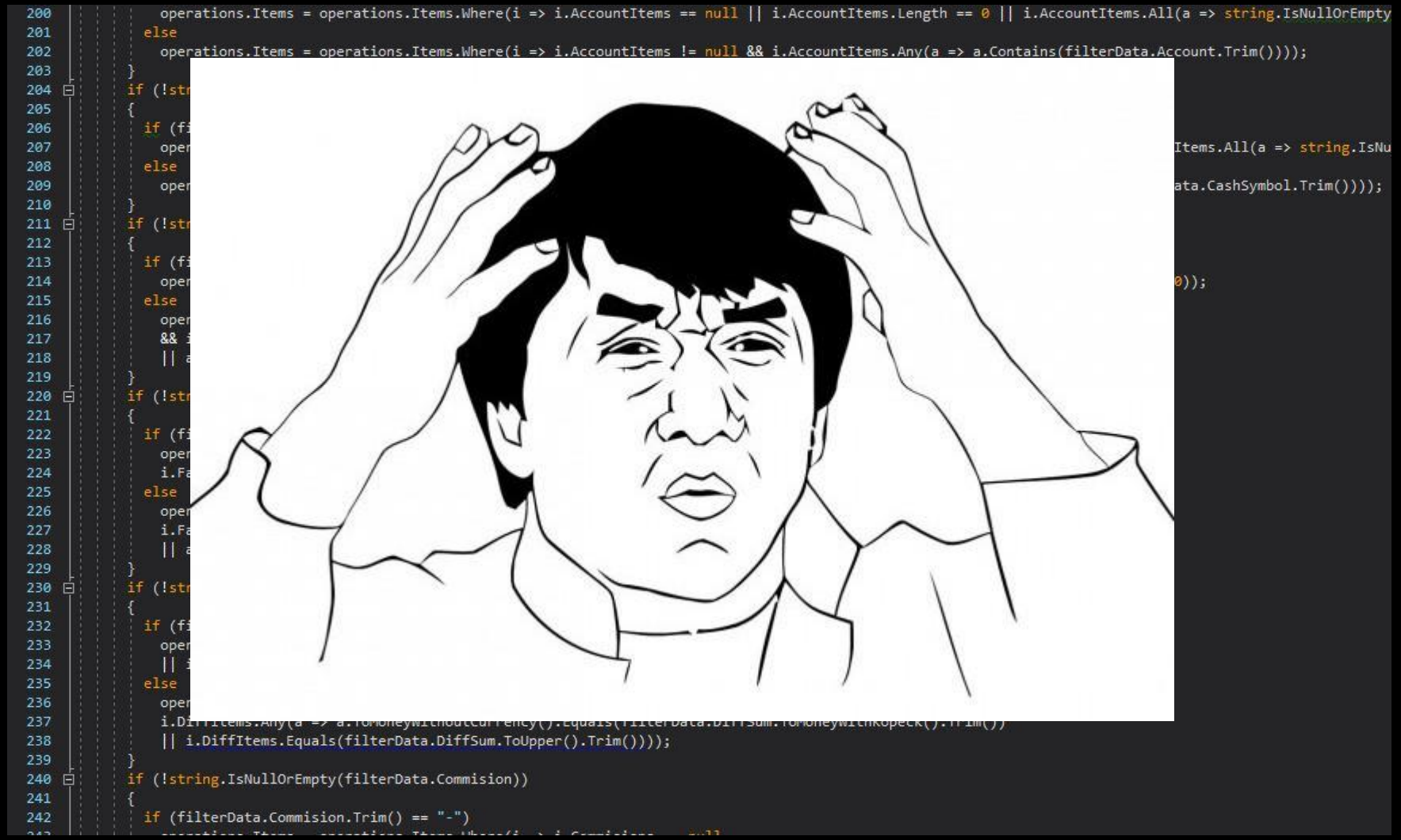
Уже с состояниями

```
public class LiftWithState
public Status Status { get; private set; }
public void GoToFloor(int floor)
{
  if (Status == Status.OnOne)
  {
    switch (floor)
    {
      case 2:
        Console.WriteLine("Go to 2 floor");
        Door(false);
        Status = Status.OnTwo;
        Door(true);
        break;
      case 3:
        Console.WriteLine("Go to 3 floor");
        Door(false);
        Status = Status.OnThree;
        Door(true);
        break;
    }
  }
}
```

```
if (Status == Status.InMotion)
{
  throw new Exception("Lift is moving!");
}
```

У нас есть лифт 2.0

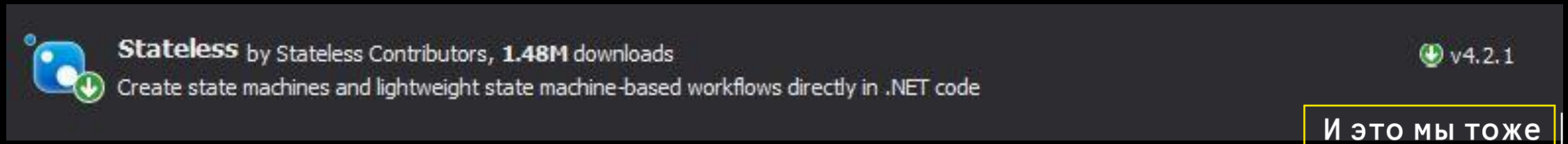
Уже с состояниями



Возможные решения

- Оставить как есть
- Написать свой конечный автомат, потому что - фатальный недостаток
- Использовать уже готовое решение

Stateless



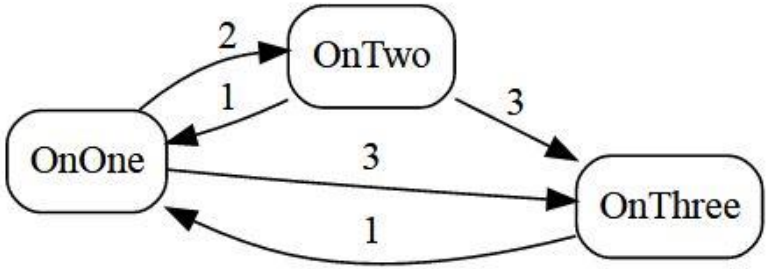
И это мы тоже добавим++

- Актуальная версия 4.2.1
- .Net Standard 1.0
- .Net Framework 4.5
- https://github.com/dotnet-state-machine/stateless

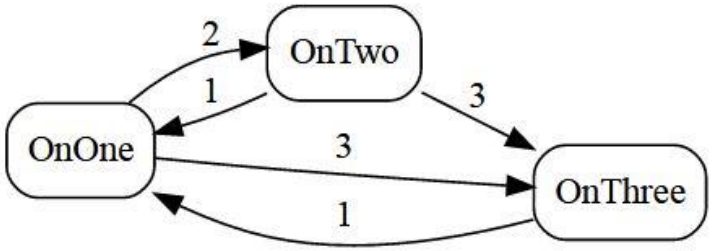
У нас есть лифт 3.0

С библиотекой Stateless

```
public class Lift
{
    private StateMachine<Status, int> _stateMachine;

    0 references | 0 exceptions
    public Status Status
    {
        get => _stateMachine.State;
    }
    public Lift()
    {
        _stateMachine = new StateMachine<Status, int>(Status.OnOne);

        _stateMachine.Configure(Status.OnOne)
                     .Permit(2, Status.OnTwo)
                     .Permit(3, Status.OnThree);
        _stateMachine.Configure(Status.OnTwo)
                     .Permit(1, Status.OnOne)
                     .Permit(3, Status.OnThree);
        _stateMachine.Configure(Status.OnThree)
                     .Permit(1, Status.OnOne);
    }
```



```
0 references | 0 exceptions
public void GoToFloor(int floor)
{
    _stateMachine.Fire(floor);
}
public string GetGraph()
{
    var umlDot = UmlDotGraph.Format(_stateMachine.GetInfo());

    return umlDot;
}
3 references | 0 exceptions
private void LiftHandler(int floor)
{
    Door(false);
    Console.WriteLine($"Go to {floor} floor");
    Door(true);
}
}
```



Хэндлеры прописываем при конфигурации состояния:

```
_stateMachine.Configure(Status.OnOne)
             .OnEntry(transition => LiftHandler(transition.Trigger))
             .OnActivate(() => Console.WriteLine("Activating!"))
             .OnExit(() => Console.WriteLine("Exit!"))
             .OnDeactivate(() => Console.WriteLine("Deactivating!"))
             .OnEntryFrom(3, () => Console.WriteLine("Entry from 3 floor!"));
```

И еще немного фич в Stateless

Райффайзен

- Substate

```
_stateMachine.Configure(Status.OnOne)
             .SubstateOf(Status.OnStandBy)
```

- Триггер с параметром

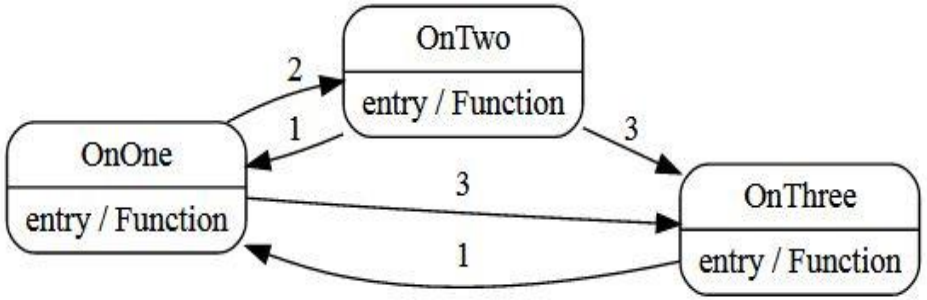
```
var setLiftHeightTrigger = _stateMachine.SetTriggerParameters<int>(1);
_stateMachine.Configure(Status.OnOne)
             .SubstateOf(Status.OnStandBy)
             .InternalTransition<int>(setLiftHeightTrigger,
                                      (height, transition) =>
                                        Console.WriteLine($"Set lift height to {height}"))

_stateMachine.Fire(setLiftHeightTrigger, 10);
```

- Re-entry

```
.PermitReentry(1)
```

Язык DOT

```
digraph {
    node [shape=Mrecord]
    rankdir="LR"
    OnOne [label="OnOne|entry / Function"];
    OnTwo [label="OnTwo|entry / Function"];
    OnThree [label="OnThree|entry / Function"];

    OnOne -> OnTwo [style="solid", label="2"];
    OnOne -> OnThree [style="solid", label="3"];
    OnTwo -> OnOne [style="solid", label="1"];
    OnTwo -> OnThree [style="solid", label="3"];
    OnThree -> OnOne [style="solid", label="1"];
}
```



var umlDot = UmlDotGraph.Format(_stateMachine.GetInfo());

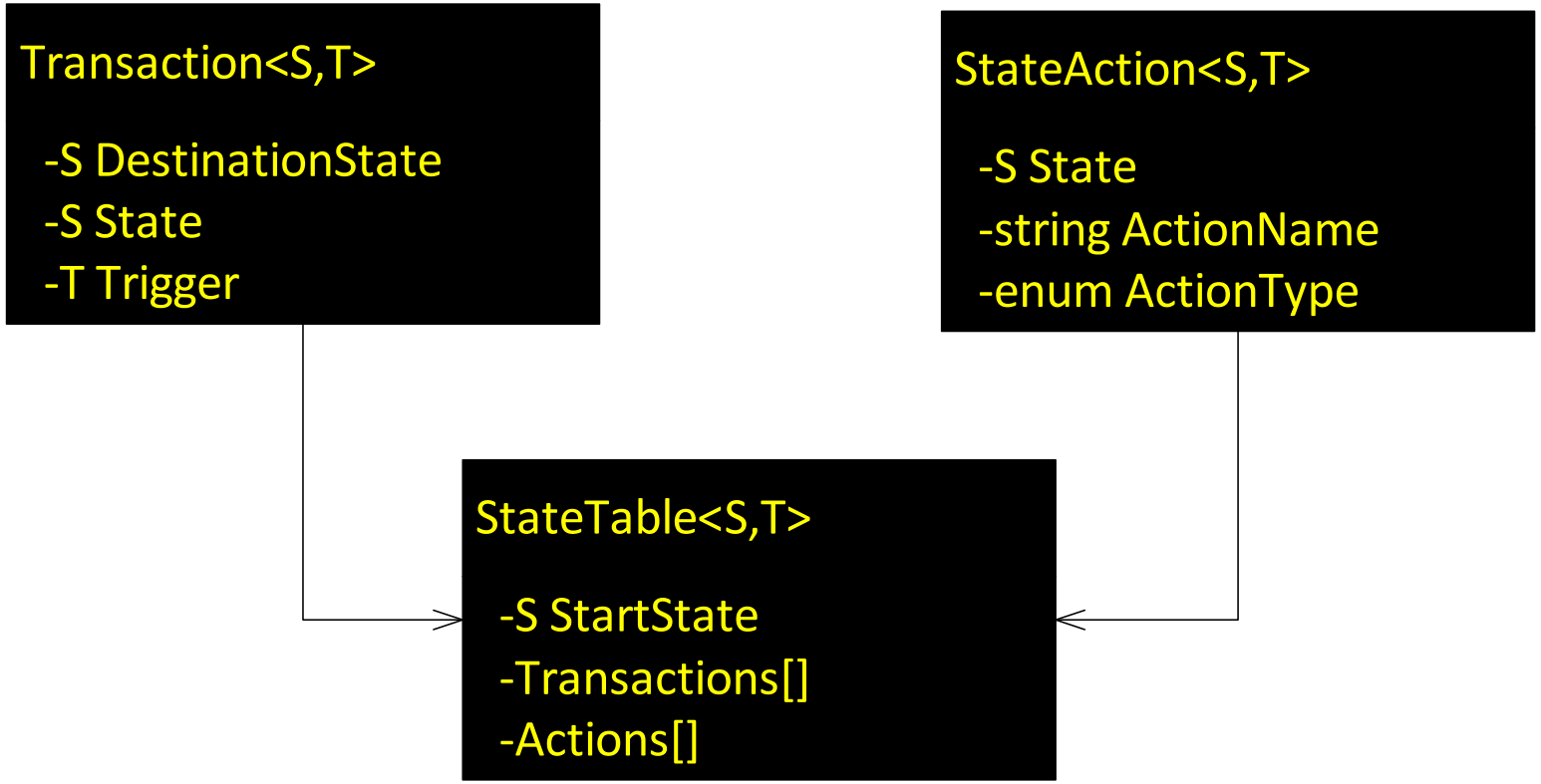
Stateless Decorator

```
public sealed class StatelessDecorator<O, S, T> where O : IStatus<S>
{
    0 references | 0 exceptions
    public StatelessDecorator(O statelessObject, StateMachine<S,T> machine)
    {
        this.StatelessObject = statelessObject;
        this.StateMachine = machine;
    }

    2 references | 0 exceptions
    public O StatelessObject { get; }
    4 references | 0 exceptions
    private StateMachine<S, T> StateMachine { get; set; }

    0 references | 0 exceptions
    public void ChangeStatus(T trigger)
    {
        StateMachine.Fire(trigger);
        this.StatelessObject.Status = StateMachine.State;
    }

    0 references | 0 exceptions
    public string GetUmlGraph()
    {
        var umlDot = UmlDotGraph.Format(this.StateMachine.GetInfo());
        return umlDot;
    }
}
```

S – Состояния
T – Триггеры
O – тип декорируемого объекта

```
public interface IStatus<S>
{
    1 reference | 0 exceptions
    S Status { get; set; }
}
```

И еще немного архитектуры. Таблицы переходов.



Десериализация.

```
public class TransactionDeserializer<O, S, T> where O : IStatus<S>
{
    1 reference | 0 exceptions
    public StatelessDecorator<O, S, T> Deserialize(O o, StateTable<S, T> stateTable)
    {
        StateMachine<S,T> stateMachine = new StateMachine<S, T>(stateTable.StartState);

        stateMachine = AddTransaction(stateMachine, stateTable.Transactions);
        stateMachine = AddActions(stateMachine, stateTable.Actions);

        StatelessDecorator<O,S,T> decorator = new StatelessDecorator<O, S, T>(o, stateMachine);

        return decorator;
    }
```

Десериализация.

```
private StateMachine<S, T> AddTransaction(StateMachine<S, T> machine, Transaction<S, T>[] transactions)
{
  foreach (var transaction in transactions)
  {
    machine.Configure(transaction.State)
           .Permit(transaction.Trigger, transaction.DestinationState);
  }

  return machine;
}

1 reference | 0 exceptions
private StateMachine<S, T> AddActions(StateMachine<S, T> machine, StateAction<S,T>[] actions)
{
  foreach (var action in actions)
  {
    switch (action.ActionType)
    {
      case ActionType.OnEntry:
        machine.Configure(action.State).OnEntry(action.Action);
        break;
      case ActionType.OnExit:
        machine.Configure(action.State).OnExit(action.Action);
        break;
    }
  }

  return machine;
}
}
```

Спасибо
за внимание!

shakirov.vr@yandex.ru
vk.com/vr.shakirov

Владислав
Шакиров

Разработчик

Райффайзен