объектно ориентированное программирование

поведенческие паттерны (2)

команда

представление какой-либо операции
в виде объекта, её выполняющего

команда

структура

команда

- команда
  интерфейс + реализации
  реализации выполняют различные операции

- отправитель
  объекты, создающие конкретные команды

- получатель
  объект(ы), ответственные за выполнение команд

проблематика

команда

- большое количество ответственности на каких-то сущностях
- необходимость разделять инициирование и выполнение действия
- дубликация логики выполнения операций

проблематика

команда

```
public sealed class FilterNode : IPipelineNode
{
    public async Task<NodeExecutionResult> ExecuteAsync(
        JsonDocument input,
        IPipelinePresentationManager presentationManager)
    {
        if (input is JsonObjectDocument objectDocument)
        {
            NodeFilterResult result = FilterSingleNode(objectDocument);

            ...
        }

        if (input is JsonArrayDocument arrayDocument)
        {
            foreach (JsonObjectDocument obj in arrayDocument.Values)
            {
                NodeFilterResult result = FilterSingleNode(obj);

                …
            }

        }
    }
}
```

решение

команда

- выделить общий интерфейс для операций над документами
- переиспользовать одни команды в других
- вызывать команды/возвращать команды в сущностях

решение

команда

```
public interface INodeCommand
{
    NodeCommandExecutionResult Execute(JsonDocument document);
}


public sealed class FilterObjectNodeCommand : INodeCommand
{
    public NodeCommandExecutionResult Execute(JsonDocument document)
    {
        ...
    }
}
```

```
public sealed class FilterArrayNodeCommand : INodeCommand
{
    private readonly FilterObjectNodeCommand _objectCommand;

    public NodeCommandExecutionResult Execute(JsonDocument document)
    {
        if (document is JsonArrayDocument array)
        {
            foreach (JsonObjectDocument value in array.Values)
            {
                var result = _objectCommand.Execute(value);
            }
        }
    }
}
```

решение

команда

```
public sealed class FilterNode : IPipelineNode
{
    public async Task<NodeExecutionResult> ExecuteAsync(
        JsonDocument input,
        IPipelinePresentationManager presentationManager)
    {
        if (input is JsonObjectDocument objectDocument)
        {
            var command = new FilterObjectNodeCommand();
            NodeCommandExecutionResult result = command.Execute(objectDocument);

            ...
        }

        if (input is JsonArrayDocument arrayDocument)
        {
            var command = new FilterArrayNodeCommand();
            NodeCommandExecutionResult result = command.Execute(arrayDocument);

            ...
        }
    }
}
```

итератор

вынесение логики прохода по коллекциям / сложным объектным структурам в отдельные типы

итератор

структура

итератор

- итератор
  интерфейс + реализации
  объекты, инкапсулирующие логику обхода коллекций/объектных структур

- “коллекция”
  какая-либо объектная структура, создающая конкретный итератор для своего обхода

проблематика

итератор

- сильная связанность объектной структуры и различной логики, подразумевающей её обход
  - либо логика сильно завязывается на детали реализации структуры
  - либо в реализацию структуры начинают протекать детали реализации логики
- низкая гибкость при реализации и (или) изменении логики обхода структуры
- дубликация логики обхода структур

проблематика

итератор

```
public interface IFileSystemComponent
{
    string Name { get; }
}

public class FileFileSystemComponent : IFileSystemComponent
{
    public string Name { get; }
}

public class DirectoryFileSystemComponent : IFileSystemComponent
{
    public string Name { get; }
    public IReadOnlyCollection<IFileSystemComponent> Components { get; }
}
```

проблематика

итератор

```
public class FileFilter
{
    public IEnumerable<IFileSystemComponent> FilterFiles(
        IFileSystemComponent component,
        string suffix)
    {
        var visitor = new FilterVisitor();
        component.Accept(visitor);

        return visitor.AllFiles.Where(file ⇒ file.Name.EndsWith(suffix));
    }

    private class FilterVisitor : IFileSystemComponentVisitor
    {
        public IReadOnlyCollection<FileFileSystemComponent> AllFiles { get; }

        public void Visit(FileFileSystemComponent component) { ... }

        public void Visit(DirectoryFileSystemComponent component) { ... }
    }
}
```

решение

итератор

- выделить отдельный интерфейс для обхода структуры
- реализовать итераторы для переиспользования логики

решение

итератор

```
public interface IEnumerator<T>
{
    bool MoveNext();

    T Current { get; }

    void Reset();
}
```

```
public class FileIterator : IFileSystemComponentVisitor, IEnumerator<FileFileSystemComponent>
{
    private readonly Queue<FileFileSystemComponent> _returnQueue = [];
    private readonly Queue<DirectoryFileSystemComponent> _iterateQueue = [];

    private readonly IFileSystemComponent _rootComponent;

    private FileFileSystemComponent? _current;

    public FileFileSystemComponent Current
        => _current ?? throw new InvalidOperationException("No current file");

    public bool MoveNext() { ... }

    public void Reset() { ... }

    public void Visit(FileFileSystemComponent component) { ... }

    public void Visit(DirectoryFileSystemComponent component) { ... }
}
```

решение

итератор

```
public class FileIterator : IFileSystemComponentVisitor, IEnumerator<FileFileSystemComponent>
{
    ...

    public FileIterator(IFileSystemComponent component)
    {
        _rootComponent = component;
        Reset();
    }

    ...

    public void Reset()
    {
        _returnQueue.Clear();
        _iterateQueue.Clear();

        _rootComponent.Accept(this);
    }

    public void Visit(FileFileSystemComponent component)
    {
        _returnQueue.Enqueue(component);
    }

    public void Visit(DirectoryFileSystemComponent component)
    {
        _iterateQueue.Enqueue(component);
    }
}
```

решение

итератор

```
public class FileIterator : IFileSystemComponentVisitor, IEnumerator<FileFileSystemComponent>
{
    ...

    public bool MoveNext()
    {
        FileFileSystemComponent? returnValue;

        while (_returnQueue.TryDequeue(out returnValue) is false
               && _iterateQueue.TryDequeue(out DirectoryFileSystemComponent? iterateValue))
        {
            foreach (IFileSystemComponent component in iterateValue.Components)
            {
                component.Accept(this);
            }
        }

        _current = returnValue;
        return returnValue is not null;
    }

    ...
}
```

решение

итератор

```
public class FileEnumerable : IEnumerable<FileFileSystemComponent>
{
    private readonly IFileSystemComponent _component;

    public FileEnumerable(IFileSystemComponent component)
    {
        _component = component;
    }

    public IEnumerator<FileFileSystemComponent> GetEnumerator() => new FileIterator(_component);

    IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
}
```

решение

итератор

```
public class FileFilter
{
    public IEnumerable<IFileSystemComponent> FilterFiles(
        IFileSystemComponent component,
        string suffix)
    {
        var fileEnumerable = new FileEnumerable(component);
        return fileEnumerable.Where(file ⇒ file.Name.EndsWith(suffix));
    }
}
```

СНИМОК

инкапсуляция логики сохранения/
восстановления состояния объекта

СНИМОК

структура

снимок

- создатель / originator
  сущность, состояние которой мы сохраняем/восстанавливаем

- снимок
  объект, представляющий состояние сущности
  снимки создаёт originator

- опекун / caretaker
  сущность, управляющая жизненным циклом создателя
  инициирует создание снимков, хранит их, инициирует восстановление снимков

проблематика

снимок

- необходимость “откатывать” состояние некоторой сущности
- сильная связанность логики сохранения/восстановления состояния
  - внешняя логика завязывается на внутреннюю структуру сущности
- лишняя мутабельность для восстановления состояния
  - сущность должна предоставлять возможность изменять все свои данные

проблематика

снимок

```
public class TextField
{
    public string Value { get; set; } = string.Empty;

    public void AddValue(string value)
    {
        Value += value.Trim();
    }
}
```

```
public class TextFieldHistory
{
    private readonly TextField _field = new();
    private readonly Stack<string> _values = [];

    public void AddValue(string value)
    {
        _values.Push(_field.Value);
        _field.AddValue(value);
    }

    public void RollbackLastOperation()
    {
        if (_values.TryPop(out string? value))
        {
            _field.Value = value;
        }
    }
}
```

проблематика

снимок

```
public class TextField
{
    public string Value { get; set; } = string.Empty;

    public int Version { get; set; }

    public void AddValue(string value)
    {
        Value += value.Trim();
        Version++;
    }
}
```

решение

снимок

- инкапсулировать логику сохранения данных в самой сущности
- инкапсулировать логику восстановления данных в самой сущности
- в хранящей сущности завязываться на сами снимки, а не на данные

решение

снимок

```
public sealed record TextFieldSnapshot(string Value, int Version);


public class TextField
{
    public string Value { get; private set; } = string.Empty;

    public int Version { get; private set; }

    public void AddValue(string value)
    {
        Value += value.Trim();
        Version++;
    }

    public TextFieldSnapshot CreateSnapshot()
    {
        return new TextFieldSnapshot(Value, Version);
    }

    public void RestoreSnapshot(TextFieldSnapshot snapshot)
    {
        Value = snapshot.Value;
        Version = snapshot.Version;
    }
}
```

```
public class TextFieldHistory
{
    private readonly TextField _field = new();
    private readonly Stack<TextFieldSnapshot> _snapshots = [];

    public void AddValue(string value)
    {
        _snapshots.Push(_field.CreateSnapshot());
        _field.AddValue(value);
    }

    public void RollbackLastOperation()
    {
        if (_snapshots.TryPop(out TextFieldSnapshot? snapshot))
        {
            _field.RestoreSnapshot(snapshot);
        }
    }
}
```

состояние

выделение условной логики относительно состояния сущности в отдельные типы

состояние

структура

состояние

- состояние
  интерфейс + реализации
  представляют собой логику сущности в различных состояниях
  также реализуют логику допустимости переходов (машина состояний)

- ядро сущности
  набор данных и поведений
  предоставляет инварианты изменения данных
  предоставляет возможность изменить своё состояние

- сущность
  фасад над ядром сущности, связывает состояния и ядро сущности

проблематика

состояние

- наличие условной логики относительно состояния сущности
- низкая гибкость изменения логики при наивной реализации
- низкая гибкость при переходах между состояниями
- низкая наглядность доступности действий относительно состояний

проблематика

состояние

```
public enum SubmissionState
{
    Active,
    Inactive,
    Completed,
    Banned,
}
```

проблематика

состояние

```
public class Submission
{
    public SubmissionState State { get; private set; }

    public DateTimeOffset? CompletedAt { get; private set; }

    public Points? Points { get; private set; }

    public bool TryActivate()
    {
        if (State is not SubmissionState.Inactive)
            return false;

        State = SubmissionState.Active;
        return true;
    }

    public bool TryDeactivate()
    {
        if (State is not SubmissionState.Active)
            return false;

        State = SubmissionState.Inactive;
        return true;
    }

    ...
}
```

проблематика

состояние

```
public class Submission
{
    ...

    public bool TryRate(Points points, DateTimeOffset timestamp)
    {
        if (State is not SubmissionState.Active)
            return false;

        CompletedAt = timestamp;
        Points = points;
        State = SubmissionState.Completed;

        return true;
    }

    public bool TryBan()
    {
        if (State is SubmissionState.Banned)
            return false;

        Points = null;
        State = SubmissionState.Banned;

        return true;
    }
}
```

решение

состояние

- выделить абстракцию для состояния
- отделить ядро сущности и фасад сущности
- перенести реализацию логики в ядро сущности
- перенести проверки доступности действий в состояния

решение

состояние

```
public class SubmissionCore
{
    public SubmissionCore(ISubmissionState state)
    {
        State = state;
    }

    public DateTimeOffset? CompletedAt { get; private set; }

    public Points? Points { get; private set; }

    public ISubmissionState State { get; private set; }

    public void UpdateState(ISubmissionState state)
    {
        State = state;
    }

    public void Rate(Points points, DateTimeOffset timestamp)
    {
        CompletedAt = timestamp;
        Points = points;
    }

    public void Ban()
    {
        Points = null;
    }
}
```

решение

состояние

```
public interface ISubmissionState
{
    public bool TryActivate(
        SubmissionCore submission);

    public bool TryDeactivate(
        SubmissionCore submission);

    public bool TryRate(
        SubmissionCore submission,
        Points points,
        DateTimeOffset timestamp);

    public bool TryBan(
        SubmissionCore submission);
}
```

```
public sealed class ActiveSubmissionState : ISubmissionState
{
    public bool TryActivate(SubmissionCore submission) => false;

    public bool TryDeactivate(SubmissionCore submission)
    {
        submission.UpdateState(new InactiveSubmissionState());
        return true;
    }

    public bool TryRate(
        SubmissionCore submission,
        Points points,
        DateTimeOffset timestamp)
    {
        submission.Rate(points, timestamp);
        submission.UpdateState(new CompletedSubmissionState());

        return true;
    }

    public bool TryBan(SubmissionCore submission)
    {
        submission.Ban();
        submission.UpdateState(new BannedSubmissionState());

        return true;
    }
}
```

решение

состояние

```
public sealed class CompletedSubmissionState : ISubmissionState
{
    public bool TryActivate(SubmissionCore submission) => false;

    public bool TryDeactivate(SubmissionCore submission) => false;

    public bool TryRate(
        SubmissionData submission,
        Points points,
        DateTimeOffset timestamp)
    {
        return false;
    }

    public bool TryBan(SubmissionCore submission)
    {
        submission.Ban();
        submission.UpdateState(new BannedSubmissionState());

        return true;
    }
}
```

решение

состояние

```
public class Submission
{
    private readonly SubmissionCore _submission;

    public Submission(SubmissionCore data)
    {
        _submission = submission;
    }

    public DateTimeOffset? CompletedAt ⇒ _submission.CompletedAt;

    public Points? Points ⇒ _submission.Points;

    public bool TryActivate()
        ⇒ _submission.State.TryActivate(_submission);

    public bool TryDeactivate()
        ⇒ _submission.State.TryDeactivate(_submission);

    public bool TryRate(Points points, DateTimeOffset timestamp)
        ⇒ _submission.State.TryRate(_submission, points, timestamp);

    public bool TryBan()
        ⇒ _submission.State.TryBan(_submission);
}
```

уточнения

состояние

- это не единственно правильный способ реализовать данный паттерн
  - можно вынести объект состояния в фасад сущности
  - ядро сущности может быть ответственно за переходы между состояниями
  - можно не разделять ядро сущности и его фасад