МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Технологии разработки программно-информационных систем

Отчет по практическому заданию № 5

Рефакторинг кода

Вариант 6

Выполнил студент группы М3О-108СВ-25

Давыдов А.П.

Принял:

Титов Ю.П.

Москва, 2025

Задание

При выполнении практической работы студенты учатся проводить рефакторинг существующего кода, кода программного обеспечения, созданного в результате выполнения творческой работы. Отчетностью по данной практической работе служит измененный код с указанием «было-стало». Для каждого изменения дается словесное описание причин изменения и достоинств, полученных в результате изменения кода. Также требуется дать словесную (числовую, приблизительную оценку) технического долга для функции/объекта/модуля, в которых произведен рефакторинг. При изучении рефакторинга важно рассмотреть различные варианты изменения кода. Так как программное обеспечение, написанное студентами, обычно не обладает сложными зависимостями, то студенты будут стремиться выполнить в результате рефакторинга только простые методы по типу «Переименование метода (Rename Method)». В результате рефакторинга студентам разрешается брать способы только из тех, что приведены в вариантах. При этом необязательно использовать все способы рефакторинга, а можно многократно повторить один из них.

Вариант 6

- Удаление управляющего флага (Remove Control Flag).
- Инкапсуляция коллекции (Encapsulate Collection).
- Разделение запроса и модификатора (Separate Query from Modifier).
- Замена конструктора фабричным методом (Replace Constructor with Factory Method).

Процесс выполнения

Правила кода в творческой работе во многом трактуются использованием самого Django, из-за чего в коде отсутствуют управляющие флаги, коллекции и конструктор. В свою очередь, модификаторы и запросы также отсутствуют по причине отсутствия таковых в применяемой фреймворком парадигме MVC.

По этой причине для анализа был взят старый фрагмент кода C#, написанный под Unity и имеющий потенциал применения в будущем:

```
public abstract class EventChainReversable : MonoBehaviour
      {
          //TODO!! : REFACTOR! Код ужаса с сотней состояний в hardcode

          [SerializeField] protected VoidEventChannelSO activate;
          [SerializeField] protected VoidEventChannelSO onActivateEnd;
          [SerializeField] protected VoidEventChannelSO finish;
          [SerializeField] protected VoidEventChannelSO onFinishEnd;

          //Вся суть была в том, чтоб эти вот поля избежать. Избежать не вышло.
          [SerializeReference]
          public EventChainReversable nextJoint;
          [SerializeReference]
          public EventChainReversable prevJoint;

          [Header("DEBUG")]
          //TODO : Добавить сюда прогресс вместо сотни bool'ов
          [SerializeField]
          private bool _inActivation = false;
          [SerializeField]
          private bool _activated = false;
          [SerializeField]
          private bool _inFinishing = false;

          private UnityAction immediateFinishAfterActivation = null;

          /// <summary>
          /// Является ли этот элемент в цепи крайним по завершению?
          /// </summary>
```

```
        [SerializeField] private bool isFinishingEdge;
        /// <summary>
        /// Является ли этот элемент в цепи крайним началом это цепи?
        /// </summary>
        [SerializeField] private bool isStartingEdge;

        protected bool IsFinishingEdge { get => isFinishingEdge; }
        protected bool IsStartingEdge { get => isStartingEdge; }

        protected virtual void Awake()
        {
            immediateFinishAfterActivation = () =>
            {
                activate.OnEventRaised -= immediateFinishAfterActivation;
                _inFinishing = true;
                _inActivation = false;
                _activated = false;
                Finish();
            };

            if (onActivateEnd == null)
                isFinishingEdge = true;

            if (onFinishEnd == null)
                isStartingEdge = true;
        }

        protected virtual void OnEnable()
        {
            activate.OnEventRaised += ChainActivationEventCatch;
            finish.OnEventRaised += ChainFinishEventCatch;
        }
        protected virtual void OnDisable()
        {
            activate.OnEventRaised -= ChainActivationEventCatch;
            finish.OnEventRaised -= ChainFinishEventCatch;
            activate.OnEventRaised -= immediateFinishAfterActivation;
        }
        private void ChainActivationEventCatch()
        {
            if (!_inActivation && !_activated && !_inFinishing)
            {
```

```
                activate.OnEventRaised -= immediateFinishAfterActivation;
                Activation();
                _inActivation = true;
            }
            else
                Debug.Log("Discard Activation! Already in some state");
        }
        private void ChainFinishEventCatch()
        {
            if (!_inFinishing)
            {
                if (isFinishingEdge && !_activated && !_inFinishing
                        && !(prevJoint._inActivation && prevJoint._activated &&
prevJoint._inFinishing)
                    && !prevJoint._inFinishing)
                {
                                                                        if    (!
activate.OnEventRaised.GetInvocationList().Contains(immediateFinishAfterActivation)
)
                                                            activate.OnEventRaised  +=
immediateFinishAfterActivation;
                }
                else
                {
                    _inFinishing = true;
                    _activated = false;
                    Finish();
                }
            }
            else
                Debug.Log("Discard finish - already finishing");
        }
        protected abstract void Activation();
        protected abstract void Finish();
        protected virtual void OnActivatedFully_Internal()
        {
            _inActivation = false;
            _activated = true;
            OnActivated();
        }
        protected virtual void OnFinishedFully_Internal()
        {
```

```
            _inActivation = false;
            _activated = false;
            _inFinishing = false;
            OnFinished();
        }
        protected abstract void OnActivated();
        protected abstract void OnFinished();
}
```

Этот скрипт взят с проекта «Сердце Алтая», и являет собой специальную фрагмент системы «Каналов событий», это система контроля вызова цепочек событий.

Данный скрипт должен совершать некий прерываемый и нагнетаемый процесс. То-есть он начинается, длится, и заканчивается. Во всех трёх случаях он должен вызывать соответствующие события (Передаваемые по каналам).

После сигнала завершения он должен вернуться обратно в начало.

Задачи:

1. В данном случае предстоит сделать много рефакторов, связанных с извлечением управляющих флагов, их целых три штуки, что провоцирует излишнюю сложность системы.
2. Под «Разделение запроса и модификатора (Separate Query from Modifier).» понимаю уже стоящую в комментарии задачу:

```
    //Вся суть была в том, чтоб эти вот поля избежать. Избежать не вышло.
    [SerializeReference]
    public EventChainReversable nextJoint;
    [SerializeReference]
    public EventChainReversable prevJoint;
```

Здесь происходит рекурсивный вызов этого же класса, что можно считать запросом. Модификатор — этот же класс, и нужно выделить каналы событий в качестве главного контроллера. Их задача как раз и заключается в

том, чтобы связывать логику таким образом, чтобы скрипты ничего не знали о вызывающем кроме самого факта вызова.

Коллекций в скрипте нет, как и конструктора (Это скрипт MonoBehaviour Unity). Потому есть 4 доступных рефактора.

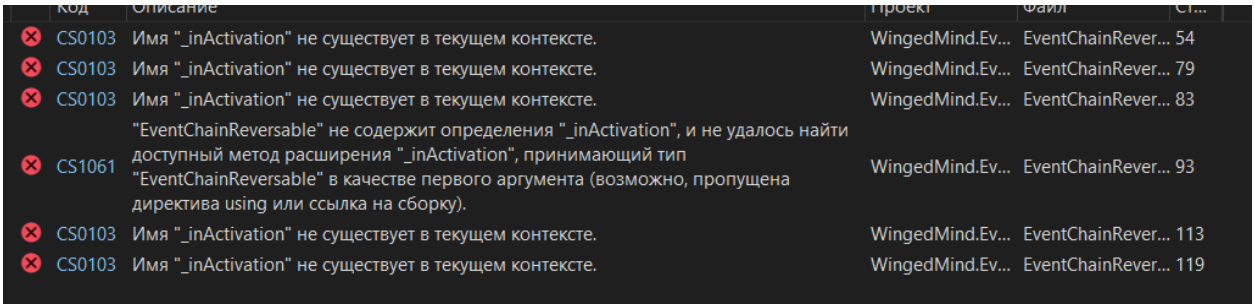
Анализ:

Во первых, здесь совершенно необходимо переименовать поля для отражения нового функционала.

```
[SerializeField] protected VoidEventChannelSO listened_raise;
[SerializeField] protected VoidEventChannelSO raiseFinish;
[SerializeField] protected VoidEventChannelSO listened_decreaseStart;
[SerializeField] protected VoidEventChannelSO decreaseFinish;
```

1) Удаление управляющего флага (Remove Control Flag).

Сначала чистка скрипта как есть от флагов:



Это оставило код практически полностью пустым.
Следует заменить сам подход и логику. Предполагается, что наследники сами будут определять прогресс и достижение конечного итога.

Тем не менее, процесс должен быть максимально автоматизирован. Система может получить сигнал начала «Нагнетания», а также сигнал его завершения.

Как минимум, нужно добавить поле, определяющее степень этого нагнетания. А также нужно поле динамики изменения этого нагнетения:

```
    [SerializeField]
    [Readonly]
    protected float progress = 0;
    [SerializeField]
    [Readonly]
    private float currentDynamic = 1;
```

Тогда наследник определяет скорость и динамику прогресса в зависимости уже от своей логики.

Итого весь функционал 5 функций перемещается в Update, где и контролируется вся логика:

```
    protected virtual void Update()
    {
        progress += currentDynamic * Time.deltaTime;

        if (progress > 0 && currentDynamic > 0)
        {
            currentDynamic = 0;
            raiseFinish.RaiseEvent();
        }
        else if (progress < 0 && currentDynamic < 0)
        {
            currentDynamic = 0;
            decreaseFinish.RaiseEvent();
        }

        progress = Mathf.Clamp01(progress);
    }
```

1.1) Разделение запроса и модификатора (Separate Query from Modifier).

В ходе предыдущего рефактора было обнаружено следующее:

```
    protected abstract void Activation();
    protected abstract void Finish();
    protected virtual void OnActivatedFully_Internal()
    {
```

```
            _activated = true;
            OnActivated();
        }
        protected virtual void OnFinishedFully_Internal()
        {
            _activated = false;
            _inFinishing = false;
            OnFinished();
        }
        protected abstract void OnActivated();
        protected abstract void OnFinished();
```

Это функции повторяющего исполнения. Кроме того, часть — избыточные запросы, и есть модификаторы на bool’ах.

Достаточно основной функции родительского функционала, и переопределённого у наследников.

```
        protected void Activate()
        {
            Activate_Internal();

            if (currentDynamic < 0)
            {
                currentDynamic = GetCurrentDynamic();
                if(currentDynamic <= 0)
                     Debug.LogError("Динамика изменений осталась отрицательной или
нулевой после сигнала о начале!");
            }
        }
        protected void BeginDecrease()
        {
            BeginDecrease_Internal();

            if (currentDynamic > 0)
            {
                currentDynamic = GetCurrentDynamic();
                if (currentDynamic >= 0)
                      Debug.LogError("Динамика изменений осталась положительной или
нулевой после сигнала о понижении!");
```

```
        }
    }
    protected abstract float GetCurrentDynamic();
    protected abstract void Activate_Internal();
    protected abstract void BeginDecrease_Internal();
```

Это одновременно упрощает систему, и решает проблему флага состояния в этой части кода.

2)

Необходимость в знании соседей `nextJoint` и `prevJoint` отпала сама собой, их использование в функциях отстутствует, а потому эти поля можно просто удалить.

Итого обновлённый код выполняет тот же функционал и поведение, но делает это гораздо проще. Убраны 3 флага и 2 смешанных поведения вида Модификация-Запрос, при этом добавлен прогресс и динамика прогресса.

```
    using UnityEngine;
    using WingedCore.Helpers.Attributes;

    namespace WingedMind.EventChannelsSystem
    {
        /// <summary>
         /// Абстрактный тип, который контроллирует цепи взаимодействий каналов событий.
          /// Скорость текущей цепи определяется наследниками и опрашивается в моменты вызовов.
        /// </summary>
        public abstract class EventChainReversable : MonoBehaviour
        {
            [SerializeField] protected VoidEventChannelSO listened_raise;
            [SerializeField] protected VoidEventChannelSO raiseFinish;
           [SerializeField] protected VoidEventChannelSO listened_decreaseStart;
            [SerializeField] protected VoidEventChannelSO decreaseFinish;

            [SerializeField]
            [Readonly]
```

```
        protected float progress = 0;
        [SerializeField]
        [Readonly]
        private float currentDynamic = 1;


        protected virtual void Update()
        {
            progress += currentDynamic * Time.deltaTime;

            if (progress > 0 && currentDynamic > 0)
            {
                currentDynamic = 0;
                raiseFinish.RaiseEvent();
            }
            else if (progress < 0 && currentDynamic < 0)
            {
                currentDynamic = 0;
                decreaseFinish.RaiseEvent();
            }

            progress = Mathf.Clamp01(progress);
        }

        protected virtual void OnEnable()
        {
            listened_raise.OnEventRaised += Activate;
            listened_decreaseStart.OnEventRaised += BeginDecrease;

        }
        protected virtual void OnDisable()
        {
            listened_raise.OnEventRaised -= Activate;
            listened_decreaseStart.OnEventRaised -= BeginDecrease;
        }

        protected void Activate()
        {
            Activate_Internal();

            if (currentDynamic < 0)
            {
```

```
                    currentDynamic = GetCurrentDynamic();
                    if (currentDynamic <= 0)
                                Debug.LogError("Динамика изменений осталась
отрицательной или нулевой после сигнала о начале!");
                }
            }
            protected void BeginDecrease()
            {
                BeginDecrease_Internal();

                if (currentDynamic > 0)
                {
                    currentDynamic = GetCurrentDynamic();
                    if (currentDynamic >= 0)
                                Debug.LogError("Динамика изменений осталась
положительной или нулевой после сигнала о понижении!");
                }
            }

            protected abstract float GetCurrentDynamic();
            protected abstract void Activate_Internal();
            protected abstract void BeginDecrease_Internal();
        }
    }
```