МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ИЖЕВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ М.Т.КАЛАШНИКОВА»
ФАКУЛЬТЕТ «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИЙ»
КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ»

ОТЧЁТ

По лабораторной работе №2
По дисциплине «Тестирование программного обеспечения»
На тему «Интеграционное тестирование»

Выполнили
студенты группы Б21-191-2 Л.М. Мирзагалямов

Проверил: Е. В. Старыгина
Старший преподаватель

Ижевск 2025

Оглавление

1. Задание

1.1. Цель работы

Целью данной лабораторной работы является разработка, реализация и тестирование системы «Управление судном», предназначенной для управления состояниями судна, обработки операций (погрузка, навигация, мониторинг) и взаимодействия с портовой инфраструктурой. Система должна обеспечивать управление состояниями, взаимодействие между объектами и выполнение ключевых сценариев использования, таких как запуск судна, погрузка/разгрузка грузов, навигация и мониторинг.

1.2. Описание предметной области

Предметная область охватывает систему управления судном в контексте кораблестроения и водного транспорта, включающую следующие основные объекты:

• **Ship (Судно)** — центральный объект системы, управляющий всеми операциями судна.

• **Captain (Капитан)** — объект, представляющий одушевленного участника, который отдает команды судну.

• **CargoSystem (Грузовая система)** — объект, ответственный за управление грузами (погрузка, разгрузка).

• **Engine (Двигатель)** — объект, управляющий состоянием двигателя судна (включение, выключение, диагностика).

• **NavigationSystem (Навигационная система)** — объект для задания маршрута и управления навигацией.

• **MonitoringSystem (Система мониторинга)** — виртуальный объект, собирающий данные о состоянии судна.

• **OperationLog (Журнал операций)** — виртуальный объект, фиксирующий все действия в системе.

• **PortDispatcher (Портовый диспетчер)** — внешний контроллер, обеспечивающий взаимодействие судна с портовой инфраструктурой.

Система поддерживает состояния судна: Останов, Ожидание, Движение, Погрузка, Разгрузка. Взаимодействия между объектами описаны через диаграммы состояний и последовательностей, включая процессы запуска, навигации, погрузки/разгрузки и мониторинга состояния.

1.3. Постановка задачи

Необходимо выполнить следующие задачи:

1. Разработать UML-диаграммы (диаграмму использования, диаграмму классов, диаграмму состояний) для системы «Управление судном».
2. Описать сценарии использования, включая запуск и остановку судна, погрузку/разгрузку грузов, навигацию, мониторинг состояния и взаимодействие с портовым диспетчером.
3. Реализовать систему на языке C#, используя принципы объектно-ориентированного программирования.
4. Провести тестирование системы с использованием фреймворков xUnit и Moq для проверки корректности переходов между состояниями и взаимодействия объектов.
5. Разместить проект на GitHub и настроить автоматическое тестирование через GitHub Actions.
6. Подготовить документацию, включающую описание предметной области, сценариев использования, состояний объектов и взаимодействия между ними.

1.4. Требования к системе

1. Система должна корректно переходить между состояниями (Останов, Ожидание, Движение, Погрузка, Разгрузка) при вызове соответствующих методов, таких как Start(), Stop(), LoadCargo(), NavigateTo().

2. Объекты системы (Ship, CargoSystem, Engine, NavigationSystem и др.) должны взаимодействовать через четко определенные методы, например TurnOn(), SetDestination(), GenerateReport().
3. Все сценарии использования должны быть покрыты тестами, включая базовые (запуск/остановка) и комбинированные (погрузка с последующей навигацией).
4. Код должен быть размещен в репозитории GitHub с настроенными автоматическими тестами через GitHub Actions.
5. Документация должна включать UML-диаграммы и подробное описание всех процессов.

2. Описание предметной области

Предметная область – управление судном.

Предметной областью данной лабораторной работы является автоматизированная система управления судном, предназначенная для выполнения операций, связанных с водным транспортом, таких как погрузка/разгрузка грузов, навигация, мониторинг состояния и взаимодействие с портовой инфраструктурой. Система моделирует процессы, характерные для кораблестроения и эксплуатации судов, обеспечивая управление состоянием судна, координацию подсистем и формирование отчётов для диспетчерских служб.

Основные компоненты системы управления судном включают:

- **Подсистемы судна:**
  - **Двигатель:** управляет движением судна, поддерживая состояния включения, выключения и диагностики неисправностей. Двигатель предоставляет данные о текущей мощности и состоянии.
  - **Грузовая система:** отвечает за погрузку и разгрузку грузов, отслеживая текущий вес и максимальную вместимость. Система имеет состояния, такие как пустая, загружается и полная.
  - **Навигационная система:** управляет маршрутом судна, задавая пункт назначения и отслеживая состояние навигации (активна/неактивна).
- **Судно:** центральный объект системы, координирующий работу подсистем (двигателя, грузовой системы, навигационной системы) и взаимодействие с внешними объектами. Судно поддерживает несколько состояний, таких как останов, ожидание, движение, погрузка и разгрузка. Оно отвечает за запуск, остановку, навигацию и операции с грузами.

- **Капитан:** объект, представляющий одушевленного участника, который отдаёт команды судну (например, навигация или запуск). Капитан имеет состояния бездействия и отдачи команд.
- **Система мониторинга:** виртуальный объект, собирающий данные о состоянии судна, двигателя и грузовой системы, формируя сводку для отчёта.
- **Журнал операций:** виртуальный объект, фиксирующий все действия в системе (например, запуск, погрузка, навигация) с временными метками для последующего анализа.
- **Портовый диспетчер:** внешний объект, обеспечивающий взаимодействие судна с портовой инфраструктурой. Диспетчер принимает запросы (например, на создание отчёта) и отправляет команды судну (запуск, погрузка, навигация).

Система работает в событийно-управляемом режиме: судно реагирует на команды капитана и портового диспетчера (запуск, остановка, запрос отчёта) и внутренние события (например, завершение погрузки). Переходы между состояниями судна описываются конечным автоматом, включающим такие состояния, как:

- **Останов:** судно неактивно, двигатель выключен.
- **Ожидание:** судно готово к выполнению команд (двигатель включён, но судно не движется).
- **Движение:** судно следует по заданному маршруту.
- **Погрузка:** выполняется загрузка грузов.
- **Разгрузка:** выполняется выгрузка грузов.

Система управления судном моделирует реальный процесс эксплуатации водного транспорта, обеспечивая надёжность операций и точность данных. Разработанная система должна быть протестирована на всех уровнях: от отдельных объектов (двигатель, грузовая система) до сценариев

использования (например, запуск → погрузка → навигация), чтобы гарантировать корректность работы и устойчивость к ошибкам.

3. Описание системы

3.1. Объект «Портовый диспетчер»

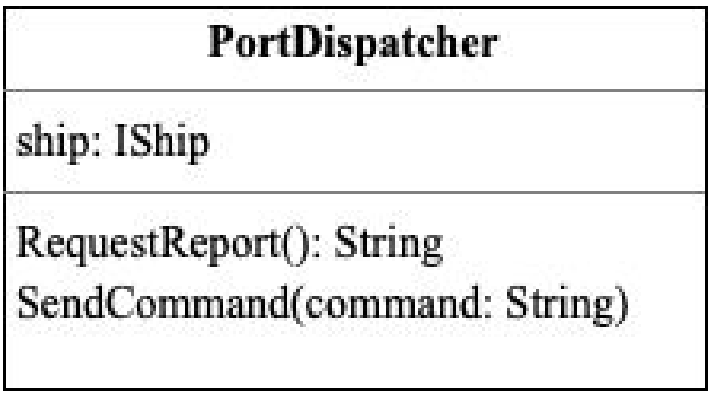

Рис. 3.1.

Внешний объект, обеспечивающий взаимодействие судна с портовой инфраструктурой.

Атрибуты:

- ship: IShip — ссылка на интерфейс судна, с которым взаимодействует диспетчер.

Методы:

- RequestReport(): String — запрашивает отчёт о состоянии судна.
- SendCommand(command: String) — отправляет команды судну (например, "start", "load", "navigate").

Состояния объекта:

- **Начальное состояние:** не определено.
- **Переход:** вызов RequestReport() инициирует запрос отчёта; вызов SendCommand() передаёт команду судну; при сбое может считаться ошибкой.
- **Завершение:** завершение запроса или выполнения команды.

3.2. Объект «Судно»

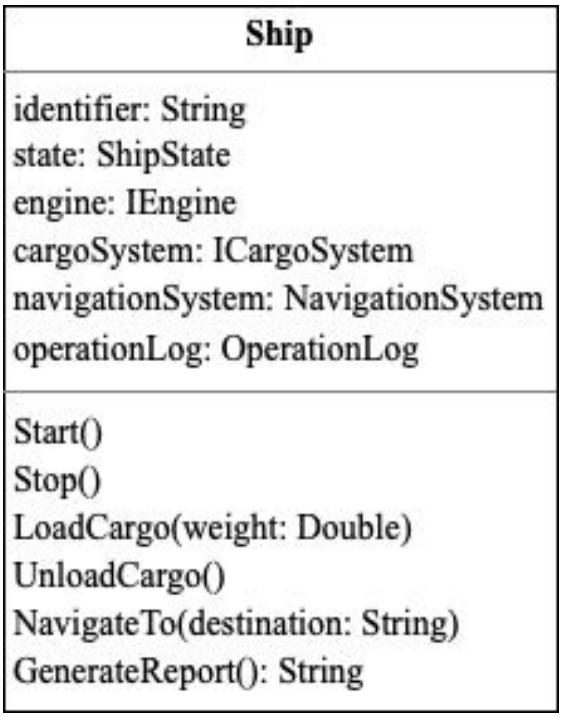

Рис. 3.2.

Центральный объект системы, координирующий работу подсистем судна.

Атрибуты:

- identifier: String {const} — уникальный идентификатор судна, задаётся при инициализации.
- state: ShipState — текущее состояние судна («останов», «ожидание», «движение», «погрузка», «разгрузка»).
- engine: IEngine — ссылка на объект двигателя.
- cargoSystem: ICargoSystem — ссылка на грузовую систему.
- navigationSystem: NavigationSystem — ссылка на навигационную систему.
- operationLog: OperationLog — ссылка на журнал операций.

Методы:

- Start() — переводит судно в состояние «ожидание».
- Stop() — останавливает судно, переводя в состояние «останов».
- LoadCargo(weight: Double) — выполняет погрузку груза.
- UnloadCargo() — выполняет разгрузку груза.
- NavigateTo(destination: String) — задаёт маршрут и переводит судно в состояние «движение».
- GenerateReport(): String — создаёт отчёт о состоянии судна.

Состояния объекта:

- **Начальное состояние:** «останов».
- **Переход:** вызов Start() переводит в «ожидание»; LoadCargo() инициирует последовательность «ожидание» → «погрузка» → «ожидание»; NavigateTo() переводит в «движение»; UnloadCargo() переводит в «разгрузка» → «ожидание»; сбой (например, попытка погрузки в неподходящем состоянии) переводит в «ошибка».
- **Завершение:** вызов Stop() возвращает в «останов».

3.3. Объект «Капитан»

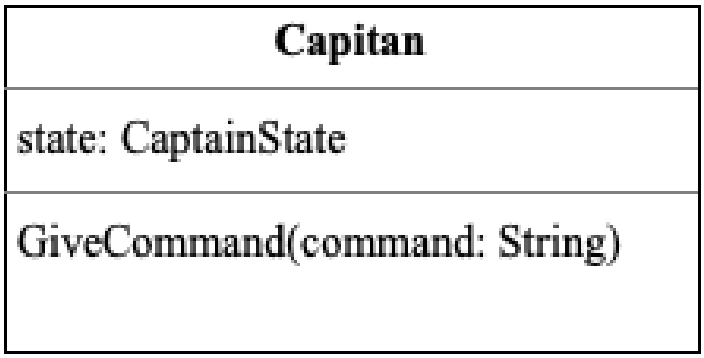

Рис. 3.3.

Реальный одушевлённый объект, представляющий капитана, отдающего команды судну.

Атрибуты:

- state: CaptainState — состояние капитана («бездействует», «командует»).

Методы:

- GiveCommand(command: String) — отдаёт команду (например, "navigate").

Состояния объекта:

- **Начальное состояние:** «бездействует».
- **Переход:** вызов GiveCommand() переводит в «командует», затем возвращает в «бездействует»; сбой (например, некорректная команда) может считаться ошибкой.
- **Завершение:** возврат в «бездействует» после выполнения команды.

3.4. Объект «Двигатель»

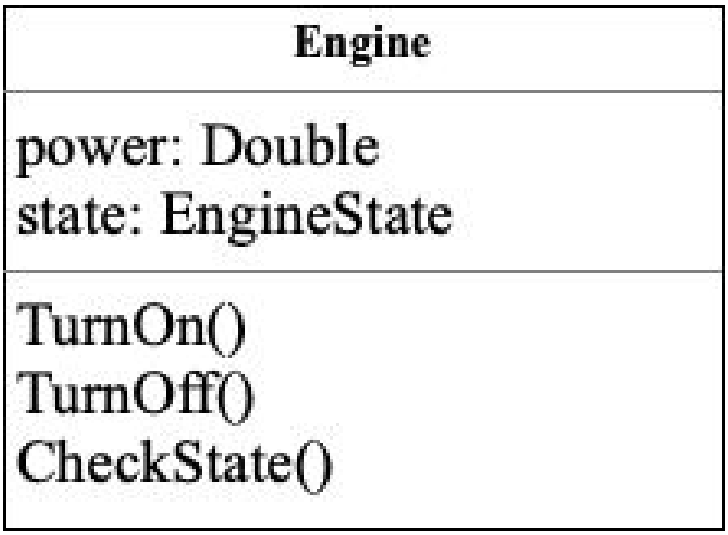

Рис. 3.4.

Реальный неодушевлённый объект для управления движением судна.

Атрибуты:

- power: Double — мощность двигателя.
- state: EngineState — состояние двигателя («выключен», «включён», «неисправен»).

Методы:

- TurnOn() — включает двигатель.
- TurnOff() — выключает двигатель.
- CheckState() — проверяет состояние двигателя (например, если мощность < 100, то «неисправен»).

Состояния объекта:

- **Начальное состояние:** «выключен».
- **Переход:** вызов TurnOn() переводит в «включён»; CheckState() может перевести в «неисправен»; TurnOff() возвращает в «выключен».
- **Завершение:** вызов TurnOff() возвращает в «выключен».

3.5. Объект «Грузовая система»

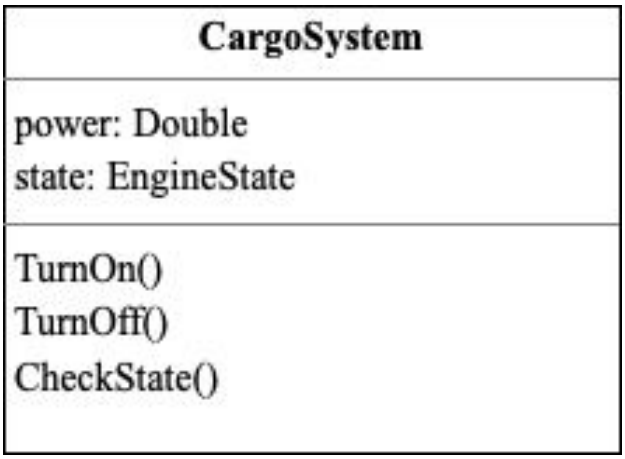

Рис. 3.5.

Реальный неодушевлённый объект для управления грузами.

Атрибуты:

- currentWeight: Double — текущий вес груза.
- maxCapacity: Double — максимальная вместимость.
- state: CargoState — состояние системы («пустая», «загружается», «полная»).

Методы:

- Load(weight: Double) — загружает груз, если вес не превышает максимальную вместимость.
- Unload() — полностью разгружает груз.

Состояния объекта:

- **Начальное состояние:** «пустая».
- **Переход:** вызов Load() переводит в «загружается», затем в «полная» (если достигнута вместимость) или «пустая»; Unload() возвращает в «пустая»; сбой (например, превышение вместимости) может считаться ошибкой.

- **Завершение:** вызов Unload() возвращает в «пустая».

3.6. Объект «Навигационная система»

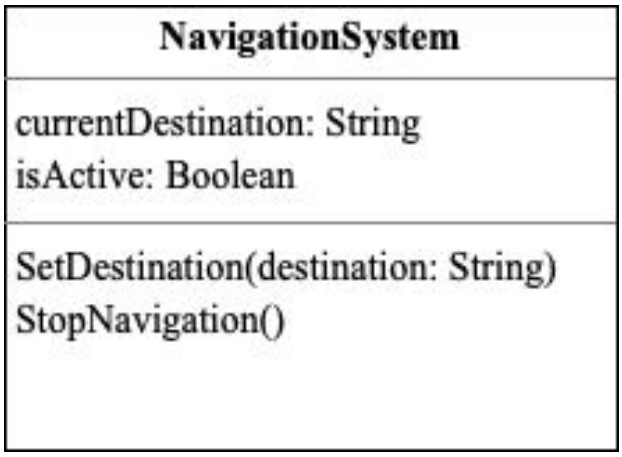

Рис. 3.6.

Реальный неодушевлённый объект для управления маршрутом судна.

Атрибуты:

- currentDestination: String — текущий пункт назначения.
- isActive: Boolean — состояние активности системы («активна», «неактивна»).

Методы:

- SetDestination(destination: String) — задаёт пункт назначения и активирует систему.
- StopNavigation() — сбрасывает пункт назначения и деактивирует систему.

Состояния объекта:

- **Начальное состояние:** «неактивна» (пункт назначения — "None").

- **Переход:** вызов SetDestination() переводит в «активна»; StopNavigation() возвращает в «неактивна».
- **Завершение:** вызов StopNavigation() возвращает в «неактивна».

3.7. Объект «Система мониторинга»

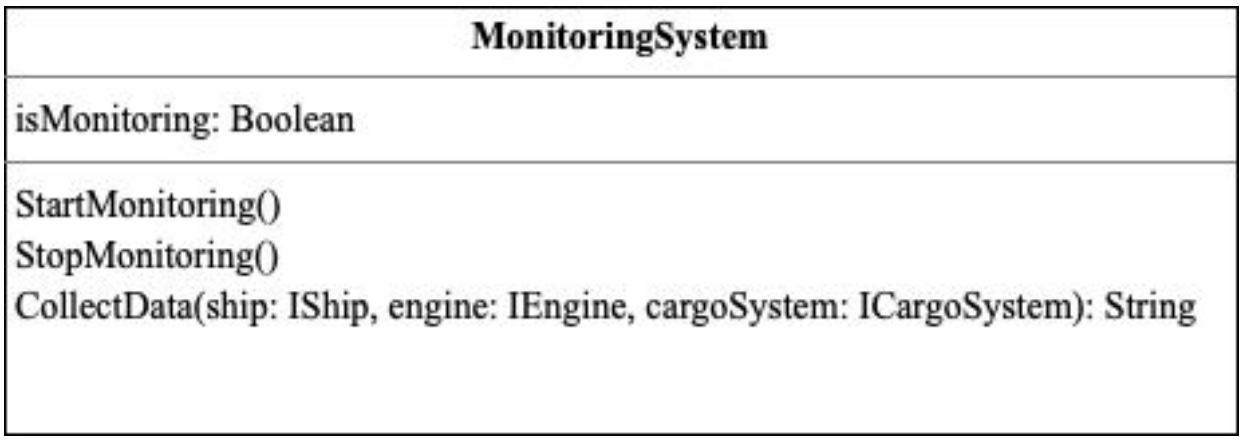

Рис. 3.7.

Виртуальный объект для сбора данных о состоянии судна.

Атрибуты:

- isMonitoring: Boolean — состояние мониторинга («активна», «неактивна»).

Методы:

- StartMonitoring() — активирует мониторинг.
- StopMonitoring() — деактивирует мониторинг.
- CollectData(ship: IShip, engine: IEngine, cargoSystem: ICargoSystem): String — собирает данные о состоянии судна, двигателя и грузовой системы.

Состояния объекта:

- **Начальное состояние:** «неактивна».

- **Переход:** вызов StartMonitoring() переводит в «активна»; CollectData() возвращает данные, если мониторинг активен; StopMonitoring() возвращает в «неактивна».
- **Завершение:** вызов StopMonitoring() возвращает в «неактивна».

3.8. Объект «Журнал операций»

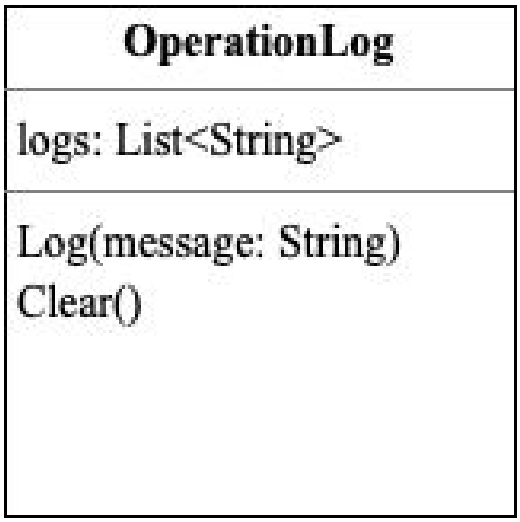

Рис. 3.8.

Виртуальный объект для логирования операций.

Атрибуты:

- logs: List<String> — список записей логов.

Методы:

- Log(message: String) — добавляет запись в лог с временной меткой.
- Clear() — очищает лог.

Состояния объекта:

- **Начальное состояние:** «пустой» (список логов пуст).

- **Переход:** вызов Log() добавляет запись, изменяя состояние списка; Clear() возвращает в «пустой».
- **Завершение:** вызов Clear() возвращает в «пустой».

3.9. Схема системы

Схема системы представлена на рисунке 3.9.

Схема системы «Управление судном»

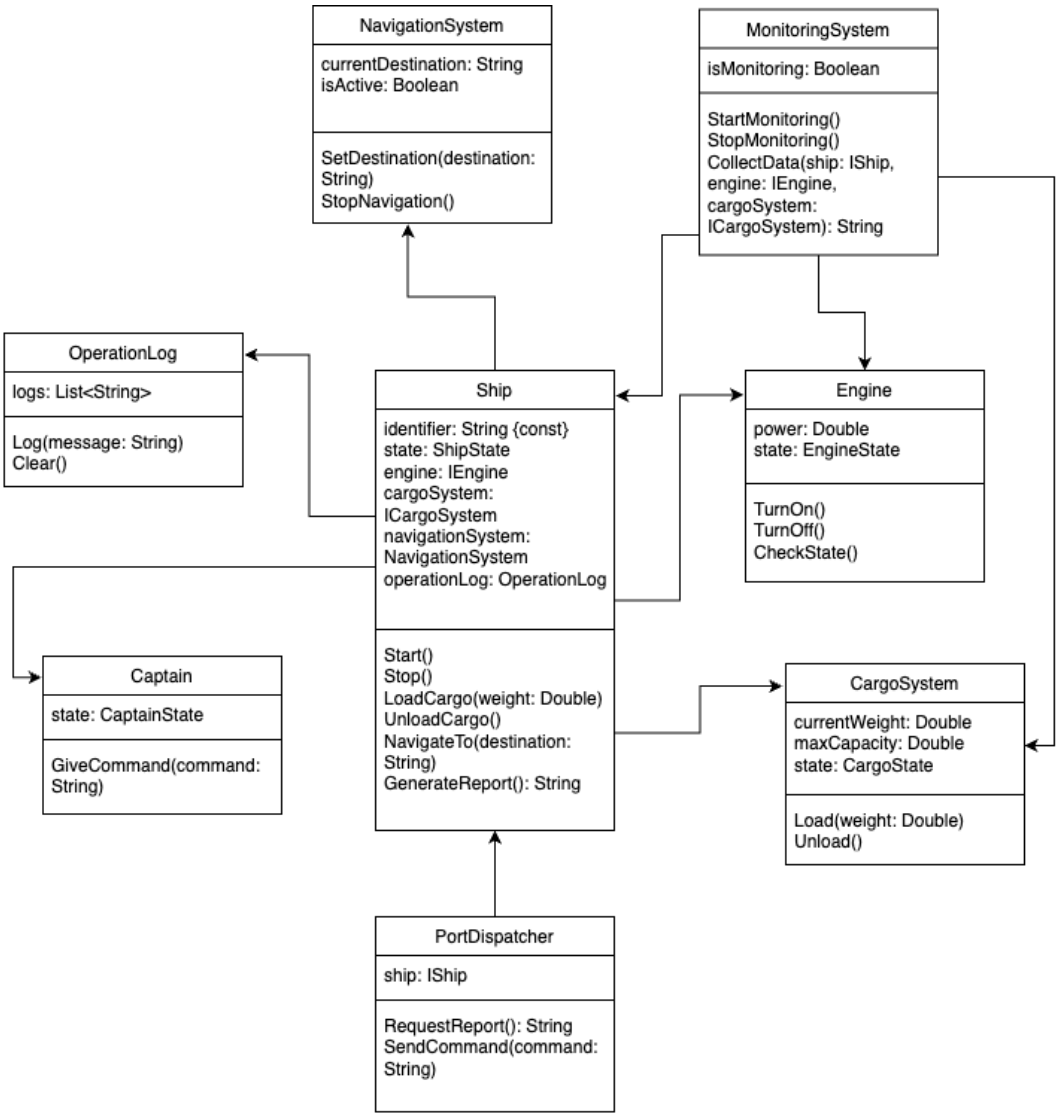

Рис. 3.9.

3.10. Описание взаимодействия между объектами

1. **Портовый диспетчер:**
    - Инициирует работу системы, запрашивая отчёты через RequestReport() или отправляя команды через SendCommand().
    - Взаимодействует с судном, передавая команды (например, "start", "navigate") и получая результаты (отчёты).
2. **Судно:**
    - Координирует работу системы, управляя состояниями и взаимодействием подсистем.
    - Принимает команды от портового диспетчера (Start(), LoadCargo(), NavigateTo()).
    - Взаимодействует с двигателем (TurnOn(), TurnOff()), грузовой системой (Load(), Unload()), навигационной системой (SetDestination()) и журналом операций (Log()).
    - Формирует отчёт через GenerateReport() для передачи диспетчеру.
3. **Капитан:**
    - Отдаёт команды через GiveCommand(), которые могут быть переданы судну (например, для навигации).
    - Взаимодействует с судном (в перспективе через интеграцию).
4. **Двигатель:**
    - Управляет движением судна через TurnOn() и TurnOff().
    - Взаимодействует с судном, изменяя своё состояние по командам (Start(), Stop()).
    - Предоставляет данные о состоянии для системы мониторинга через CollectData().
5. **Грузовая система:**

- Управляет грузами через Load() и Unload().
- Взаимодействует с судном, выполняя команды LoadCargo() и UnloadCargo().
- Предоставляет данные о текущем весе для системы мониторинга через CollectData().

6. **Навигационная система:**

   - Управляет маршрутом через SetDestination() и StopNavigation().
   - Взаимодействует с судном, выполняя команды NavigateTo().
   - Предоставляет данные о текущем пункте назначения для формирования отчёта через GenerateReport().

7. **Система мониторинга:**

   - Собирает данные о состоянии судна, двигателя и грузовой системы через CollectData().
   - Взаимодействует с судном, предоставляя сводку для отчёта.

8. **Журнал операций:**

   - Фиксирует все действия через Log().
   - Взаимодействует с судном, записывая операции (запуск, погрузка, навигация).

4. Тестирование свойств и методов объектов

4.1. Тестирование объекта «Ship»

```
using WeatherApp.Interfaces;

namespace WeatherApp.Test.UnitTests;

using Moq;
using WeatherApp.Models;

public class ShipTests
{
    private readonly Mock<IEngine> engineMock;
    private readonly Mock<ICargoSystem> cargoSystemMock;
    private readonly Mock<NavigationSystem>
navigationSystemMock;
    private readonly Mock<OperationLog> operationLogMock;
    private readonly Ship ship;

    public ShipTests()
    {
        engineMock = new Mock<IEngine>();
        cargoSystemMock = new Mock<ICargoSystem>();
        navigationSystemMock = new Mock<NavigationSystem>();
        operationLogMock = new Mock<OperationLog>();
        ship = new Ship("SHIP001", engineMock.Object,
cargoSystemMock.Object, navigationSystemMock.Object,
            operationLogMock.Object);
    }

    // Тестирование свойства
    [Fact]
    public void Identifier_ReturnsCorrectValue()
    {
        Assert.Equal("SHIP001", ship.Identifier);
    }

    [Fact]
    public void State_InitiallyStopped()
    {
        Assert.Equal(ShipState.Stopped, ship.State);
    }

    // Тестирование методов и переходов состояний
    [Fact]
    public void Start_ChangesStateToWaiting()
    {
        ship.Start();
        Assert.Equal(ShipState.Waiting, ship.State);
        engineMock.Verify(e => e.TurnOn(), Times.Once());
```

```
        operationLogMock.Verify(l => l.Log("Ship started and
moved to Waiting state."), Times.Once());
    }

    [Fact]
    public void Stop_ChangesStateToStopped()
    {
        ship.Start(); // Сначала переводим в состояние Waiting
        ship.Stop();
        Assert.Equal(ShipState.Stopped, ship.State);
        engineMock.Verify(e => e.TurnOff(), Times.Once());
    }

    [Fact]
    public void LoadCargo_ChangesStateToLoadingThenWaiting()
    {
        ship.Start(); // Переходим в Waiting
        cargoSystemMock.Setup(c =>
c.CurrentWeight).Returns(100.0);
        ship.LoadCargo(100.0);
        Assert.Equal(ShipState.Waiting, ship.State); // После
загрузки возвращается в Waiting
        cargoSystemMock.Verify(c => c.Load(100.0),
Times.Once());
    }

    [Fact]
    public void NavigateTo_ChangesStateToMoving()
    {
        ship.Start(); // Переходим в Waiting
        ship.NavigateTo("Port B");
        Assert.Equal(ShipState.Moving, ship.State);
        navigationSystemMock.Verify(n => n.SetDestination("Port
B"), Times.Once());
    }

    [Fact]
    public void GenerateReport_ReturnsCorrectReport()
    {
        cargoSystemMock.Setup(c =>
c.CurrentWeight).Returns(500.0);
        navigationSystemMock.Setup(n =>
n.CurrentDestination).Returns("Port B");
        ship.Start(); // Переходим в Waiting
        var report = ship.GenerateReport();
        Assert.Equal("Ship SHIP001: State=Waiting, Cargo=500
tons, Destination=Port B", report);
    }

    // Тестирование сценария: Останов → Ожидание → Останов
    [Fact]
    public void Scenario_StopToWaitingToStop()
```

```
    {
        Assert.Equal(ShipState.Stopped, ship.State);
        ship.Start();
        Assert.Equal(ShipState.Waiting, ship.State);
        ship.Stop();
        Assert.Equal(ShipState.Stopped, ship.State);
    }

    // Тестирование сценария: Ожидание → Погрузка → Ожидание
    [Fact]
    public void Scenario_WaitingToLoadingToWaiting()
    {
        ship.Start();
        Assert.Equal(ShipState.Waiting, ship.State);
        ship.LoadCargo(100.0);
        Assert.Equal(ShipState.Waiting, ship.State);
    }

    // Тестирование сценария: Ожидание → Движение → Ожидание
    [Fact]
    public void Scenario_WaitingToMovingToWaiting()
    {
        ship.Start();
        Assert.Equal(ShipState.Waiting, ship.State);
        ship.NavigateTo("Port B");
        Assert.Equal(ShipState.Moving, ship.State);
        ship.Wait();
        Assert.Equal(ShipState.Waiting, ship.State);
    }
}
```

4.2. Тестирование объекта «CargoSystem»

```
using WeatherApp.Models;

namespace WeatherApp.Test.UnitTests;

public class CargoSystemTests
{
    private readonly CargoSystem _cargoSystem;

    public CargoSystemTests()
    {
        _cargoSystem = new CargoSystem(1000.0);
    }

    // Тест атрибута MaxCapacity
    [Fact]
    public void MaxCapacity_ShouldReturnCorrectValue()
    {
```

```
        Assert.Equal(1000.0, _cargoSystem.MaxCapacity);
    }

    // Тест метода Load
    [Fact]
    public void Load_ShouldUpdateWeightAndState()
    {
        _cargoSystem.Load(500.0);
        Assert.Equal(500.0, _cargoSystem.CurrentWeight);
        Assert.Equal(CargoState.Empty, _cargoSystem.State);
    }

    // Тест метода Load при достижении максимальной вместимости
    [Fact]
    public void Load_AtMaxCapacity_ShouldSetStateToFull()
    {
        _cargoSystem.Load(1000.0);
        Assert.Equal(1000.0, _cargoSystem.CurrentWeight);
        Assert.Equal(CargoState.Full, _cargoSystem.State);
    }

    // Тест метода Unload
    [Fact]
    public void Unload_ShouldResetWeightAndState()
    {
        _cargoSystem.Load(500.0);
        _cargoSystem.Unload();
        Assert.Equal(0.0, _cargoSystem.CurrentWeight);
        Assert.Equal(CargoState.Empty, _cargoSystem.State);
    }

    // Тест исключения для Load при превышении вместимости
    [Fact]
    public void Load_WhenExceedsCapacity_ShouldThrowException()
    {
        Assert.Throws<InvalidOperationException>(() =>
_cargoSystem.Load(1100.0));
    }
}
```

4.3. Тестирование объекта «Engine»

```
using WeatherApp.Models;

namespace WeatherApp.Test.UnitTests;

public class EngineTests
{
    private readonly Engine engine;
```

```
    public EngineTests()
    {
        engine = new Engine(200.0);
    }

    // Тестирование свойства
    [Fact]
    public void Power_ReturnsCorrectValue()
    {
        Assert.Equal(200.0, engine.Power);
    }

    [Fact]
    public void State_InitiallyOff()
    {
        Assert.Equal(EngineState.Off, engine.State);
    }

    // Тестирование методов и состояний
    [Fact]
    public void TurnOn_ChangesStateToOn()
    {
        engine.TurnOn();
        Assert.Equal(EngineState.On, engine.State);
    }

    [Fact]
    public void TurnOff_ChangesStateToOff()
    {
        engine.TurnOn();
        engine.TurnOff();
        Assert.Equal(EngineState.Off, engine.State);
    }

    [Fact]
    public void CheckState_PowerBelow100_SetsFaulty()
    {
        var faultyEngine = new Engine(50.0);
        faultyEngine.CheckState();
        Assert.Equal(EngineState.Faulty, faultyEngine.State);
    }
}
```

4.4. Тестирование объекта «NavigationSystem»

```
using WeatherApp.Models;

namespace WeatherApp.Test.UnitTests;

public class NavigationSystemTests
{
    private readonly NavigationSystem navigationSystem;

    public NavigationSystemTests()
    {
        navigationSystem = new NavigationSystem();
    }

    // Тестирование свойств
    [Fact]
    public void CurrentDestination_InitiallyNone()
    {
        Assert.Equal("None",
navigationSystem.CurrentDestination);
    }

    [Fact]
    public void IsActive_InitiallyFalse()
    {
        Assert.False(navigationSystem.IsActive);
    }

    // Тестирование методов и состояний
    [Fact]
    public void SetDestination_UpdatesDestinationAndActivates()
    {
        navigationSystem.SetDestination("Port B");
        Assert.Equal("Port B",
navigationSystem.CurrentDestination);
        Assert.True(navigationSystem.IsActive);
    }

    [Fact]
    public void StopNavigation_ResetsDestinationAndDeactivates()
    {
        navigationSystem.SetDestination("Port B");
        navigationSystem.StopNavigation();
        Assert.Equal("None",
navigationSystem.CurrentDestination);
        Assert.False(navigationSystem.IsActive);
    }
}
```

4.5. Тестирование объекта «Capitan»

```
using WeatherApp.Models;

namespace WeatherApp.Test.UnitTests;

public class CaptainTests
{
    private readonly Captain captain;

    public CaptainTests()
    {
        captain = new Captain();
    }

    // Тестирование свойства
    [Fact]
    public void State_InitiallyIdle()
    {
        Assert.Equal(CaptainState.Idle, captain.State);
    }

    // Тестирование методов и состояний
    [Fact]
    public void GiveCommand_ChangesStateToCommandingThenIdle()
    {
        captain.GiveCommand("navigate");
        Assert.Equal(CaptainState.Idle, captain.State); // После
выполнения команды возвращается в Idle
    }
}
```

4.6. Тестирование объекта «MonitoringSystem»

```
using Moq;
using WeatherApp.Interfaces;
using WeatherApp.Models;
using Xunit;

public class MonitoringSystemTests
{
    private readonly MonitoringSystem monitoringSystem;
    private readonly Mock<IShip> shipMock;
    private readonly Mock<IEngine> engineMock;
    private readonly Mock<ICargoSystem> cargoSystemMock;

    public MonitoringSystemTests()
    {
        monitoringSystem = new MonitoringSystem();
```

```
        shipMock = new Mock<IShip>();
        engineMock = new Mock<IEngine>();
        cargoSystemMock = new Mock<ICargoSystem>();
    }

    [Fact]
    public void IsMonitoring_InitiallyFalse()
    {
        Assert.False(monitoringSystem.IsMonitoring);
    }

    [Fact]
    public void StartMonitoring_SetsIsMonitoringTrue()
    {
        monitoringSystem.StartMonitoring();
        Assert.True(monitoringSystem.IsMonitoring);
    }

    [Fact]
    public void CollectData_WhenMonitoring_ReturnsData()
    {
        shipMock.Setup(s => s.State).Returns(ShipState.Waiting);
        engineMock.Setup(e => e.State).Returns(EngineState.On);
        cargoSystemMock.Setup(c =>
c.CurrentWeight).Returns(500.0);
        monitoringSystem.StartMonitoring();
        var data = monitoringSystem.CollectData(shipMock.Object,
engineMock.Object, cargoSystemMock.Object);
        Assert.Equal("Ship State: Waiting, Engine State: On,
Cargo: 500 tons", data);
    }

    [Fact]
    public void CollectData_WhenNotMonitoring_ReturnsError()
    {
        var data = monitoringSystem.CollectData(shipMock.Object,
engineMock.Object, cargoSystemMock.Object);
        Assert.Equal("Monitoring is not active.", data);
    }
}
```

4.7. Тестирование объекта «OperationLog»

```
using WeatherApp.Models;

namespace WeatherApp.Test.UnitTests;

public class OperationLogTests
{
    private readonly OperationLog operationLog;
```

```
    public OperationLogTests()
    {
        operationLog = new OperationLog();
    }

    // Тестирование свойств
    [Fact]
    public void Logs_InitiallyEmpty()
    {
        Assert.Empty(operationLog.Logs);
    }

    // Тестирование методов
    [Fact]
    public void Log_AddsEntry()
    {
        operationLog.Log("Test message");
        Assert.Single(operationLog.Logs);
        Assert.Contains("Test message", operationLog.Logs[0]);
    }

    [Fact]
    public void Clear_EmptiesLogs()
    {
        operationLog.Log("Test message");
        operationLog.Clear();
        Assert.Empty(operationLog.Logs);
    }
}
```

4.8. Тестирование объекта «PortDispatcher»

```
using Moq;
using WeatherApp.Interfaces;
using WeatherApp.Models;

namespace WeatherApp.Test.UnitTests;

public class PortDispatcherTests
{
    private readonly Mock<IEngine> engineMock;
    private readonly Mock<ICargoSystem> cargoSystemMock;
    private readonly Mock<NavigationSystem>
navigationSystemMock;
    private readonly Mock<OperationLog> operationLogMock;
    private readonly IShip ship;
    private readonly PortDispatcher portDispatcher;

    public PortDispatcherTests()
    {
```

```
        engineMock = new Mock<IEngine>();
        cargoSystemMock = new Mock<ICargoSystem>();
        navigationSystemMock = new Mock<NavigationSystem>();
        operationLogMock = new Mock<OperationLog>();
        ship = new Ship("SHIP001", engineMock.Object,
cargoSystemMock.Object, navigationSystemMock.Object,
operationLogMock.Object);
        portDispatcher = new PortDispatcher();
    }

    // Тестирование метода RequestReport
    [Fact]
    public void RequestReport_ReturnsShipReport()
    {
        cargoSystemMock.Setup(c =>
c.CurrentWeight).Returns(500.0);
        navigationSystemMock.Setup(n =>
n.CurrentDestination).Returns("Port B");
        ship.Start();
        var report = portDispatcher.RequestReport(ship);
        Assert.Equal("Ship SHIP001: State=Waiting, Cargo=500
tons, Destination=Port B", report);
    }

    // Тестирование сценария взаимодействия
    [Fact]
    public void SendCommand_Navigate_ChangesShipState()
    {
        ship.Start(); // Переходим в Waiting
        portDispatcher.SendCommand(ship, "navigate");
        Assert.Equal(ShipState.Moving, ship.State);
        navigationSystemMock.Verify(n => n.SetDestination("Port
B"), Times.Once());
    }
}
```

5. Тестирование состояний объектов

5.1. Модель состояний системы

Приведём схему модели состояний на рисунке 5.1.

Модель состояний системы

Рис. 5.1

5.2. Описание модели состояний системы

Модель состояний системы, управляемой объектом PortDispatcher, описывает жизненный цикл судна с учётом всех ключевых процессов, включая запуск, погрузку/разгрузку грузов, навигацию и мониторинг состояния.

1. **Останов:** начальное состояние системы. Судно полностью остановлено, двигатель выключен, и система ожидает команды для запуска.

- **Переход:** вызов метода Start() переводит систему в состояние Ожидание.

2. **Ожидание:** система активна, двигатель включён, но судно не выполняет операций, ожидая дальнейших команд.
    - **Переходы:**
        - Stop() возвращает систему в Останов.
        - LoadCargo() переводит в Погрузка.
        - UnloadCargo() переводит в Разгрузка.
        - NavigateTo() переводит в Движение.
3. **Погрузка:** система выполняет погрузку грузов через метод LoadCargo().
    - **Переход:**
        - После завершения возвращается в Ожидание.
4. **Разгрузка:** система выполняет разгрузку грузов через метод UnloadCargo().
    - **Переход:**
        - После завершения возвращается в Ожидание.
5. **Движение:** система следует по заданному маршруту через метод NavigateTo().
    - **Переход:**
        - Вызов Wait() возвращает в Ожидание.

5.3. Описание последовательностей состояний для тестирования

Опишем последовательности состояний для тестирования системы управления судном:

1. **Останов → Ожидание → Останов.**
   Проверяет базовый цикл запуска и остановки системы. Тест должен убедиться, что система корректно переходит
   из Останов в Ожидание при вызове Start() и возвращается
   в Останов при вызове Stop().

2. **Ожидание → Погрузка → Ожидание.**
   Проверяет процесс погрузки грузов. Тест должен убедиться, что система корректно проходит через LoadCargo() и возвращается
   в Ожидание.

3. **Ожидание → Разгрузка → Ожидание.**
   Проверяет процесс разгрузки грузов. Тест должен убедиться, что система корректно проходит через UnloadCargo() и возвращается
   в Ожидание.

4. **Ожидание → Движение → Ожидание.**
   Проверяет процесс навигации. Тест должен убедиться, что система переходит в Движение при вызове NavigateTo(), а затем возвращается
   в Ожидание при вызове Wait().

5. **Ожидание → Погрузка → Ожидание → Движение → Ожидание.**
   Проверяет комбинированный сценарий погрузки и последующей навигации. Тест должен убедиться, что система корректно проходит через LoadCargo(), возвращается в Ожидание, затем
   выполняет NavigateTo() и снова возвращается в Ожидание через Wait().

6. **Ожидание → Разгрузка → Ожидание → Движение → Ожидание.**
   Проверяет комбинированный сценарий разгрузки и последующей навигации. Тест должен убедиться, что система корректно проходит

через UnloadCargo(), возвращается в Ожидание, затем выполняет NavigateTo() и снова возвращается в Ожидание через Wait().

7. **Ожидание → Погрузка → Ожидание → Разгрузка → Ожидание → Движение → Ожидание.**
   Проверяет полный цикл операций: погрузка, разгрузка и навигация. Тест должен убедиться, что система корректно проходит через LoadCargo(), UnloadCargo(), затем выполняет NavigateTo() и возвращается в Ожидание через Wait().

6. Тестирование сценариев и вариантов использования

6.1. Диаграмма вариантов использования

Приведём диаграмму вариантов использования на рисунке 6.1.

Диаграмма вариантов использования «Управление судном»

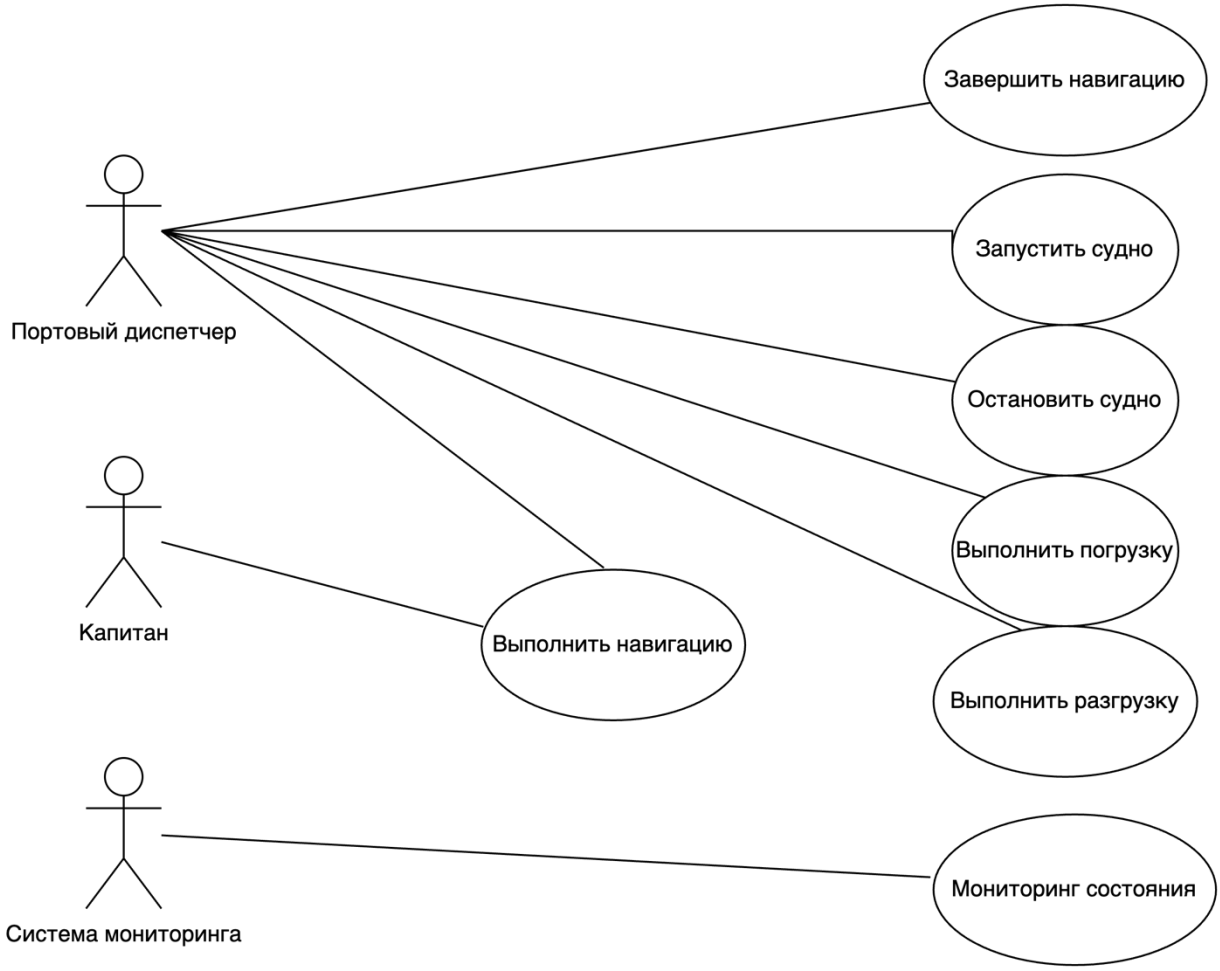

Рис. 6.1

Варианты использования

Запустить судно

Актёр: PortDispatcher

Описание: PortDispatcher вызывает метод Start(), чтобы перевести объект Ship из состояния «Останов» в «Ожидание». Это позволяет начать выполнение операций, таких как погрузка или навигация.

Предусловие: Ship находится в состоянии «Останов».

Постусловие: Ship переходит в состояние «Ожидание».

Остановить судно

Актёр: PortDispatcher

Описание: PortDispatcher вызывает метод Stop(), чтобы остановить Ship и перевести его в состояние «Останов».

Предусловие: Ship находится в состоянии «Ожидание» или другом активном состоянии.

Постусловие: Ship переходит в состояние «Останов».

Выполнить погрузку и навигацию

Актёр: PortDispatcher

Описание: PortDispatcher инициирует процесс погрузки грузов, отправляя команду Ship через метод LoadCargo(). После завершения погрузки вызывается метод NavigateTo(), чтобы перевести судно в состояние «Движение».

Предусловие: Ship находится в состоянии «Ожидание».

Постусловие: Ship проходит через состояния «Погрузка» и «Ожидание», затем переходит в «Движение».

6.2. Диаграммы последовательности

Диаграмма последовательности варианта использования «Завершить навигацию» приведена на рисунке 6.2.

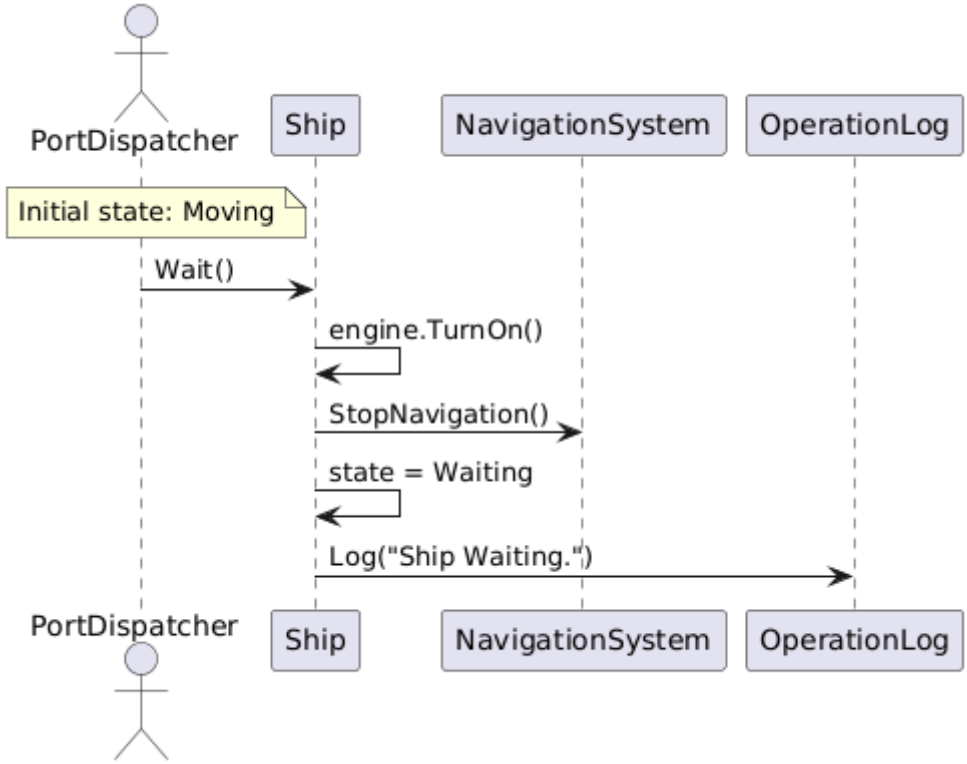

Рис. 6.2

Диаграмма последовательности «Завершить навигацию»

Диаграмма последовательности варианта использования «Запустить судно» приведена на рисунке 6.3.

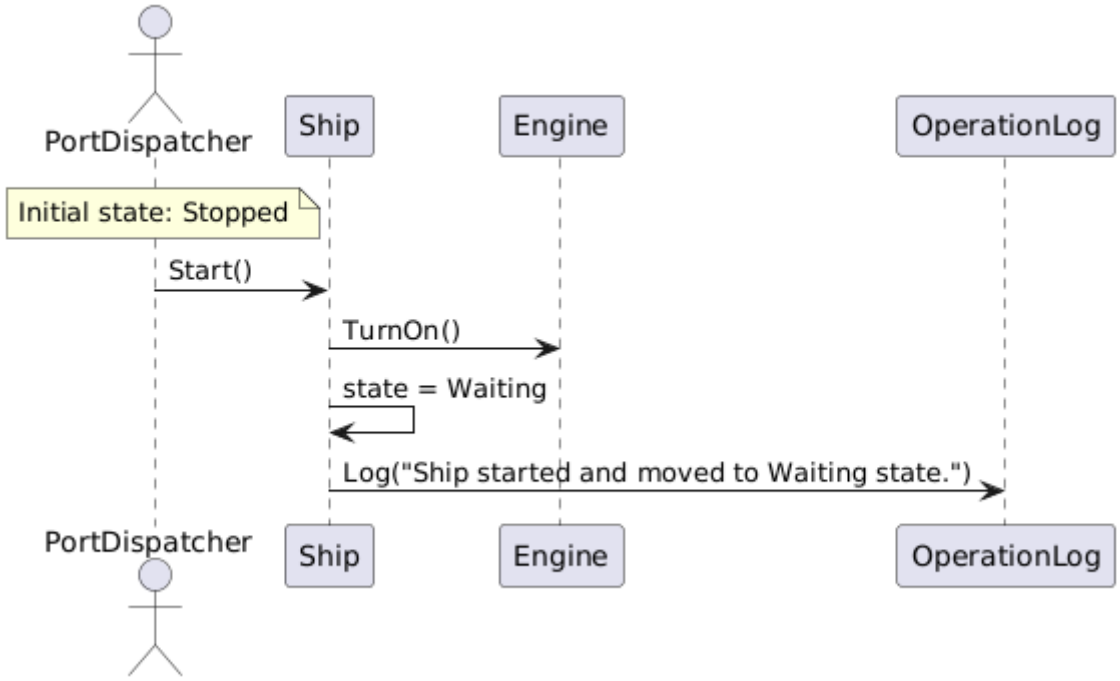

Рис. 6.3

Диаграмма последовательности «Запустить судно»

Диаграмма последовательности варианта использования «Остановить судно» приведена на рисунке 6.4.

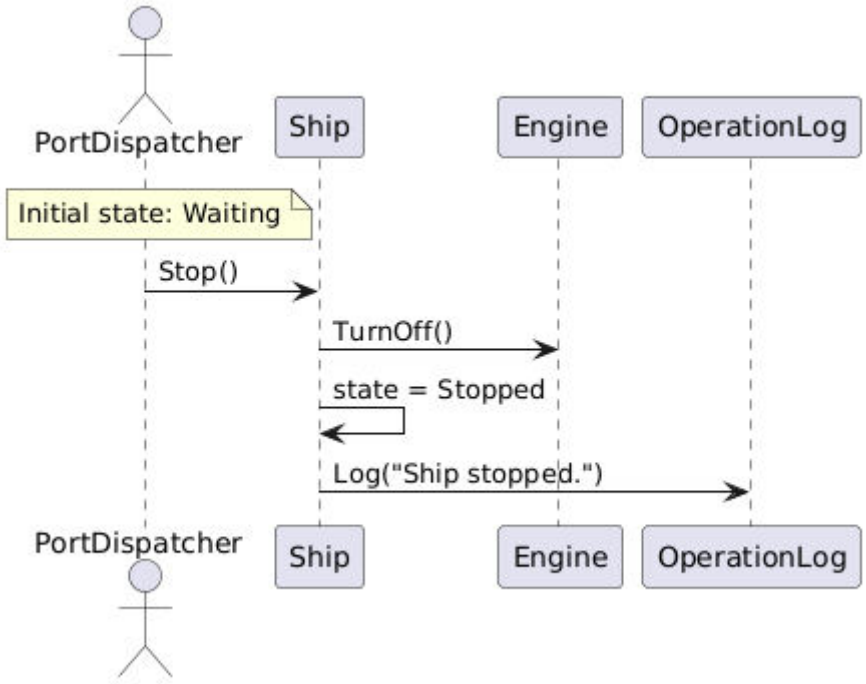

Рис. 6.4

Диаграмма последовательности «Остановить судно»

Диаграмма последовательности варианта использования «Выполнить погрузку» приведена на рисунке 6.5.

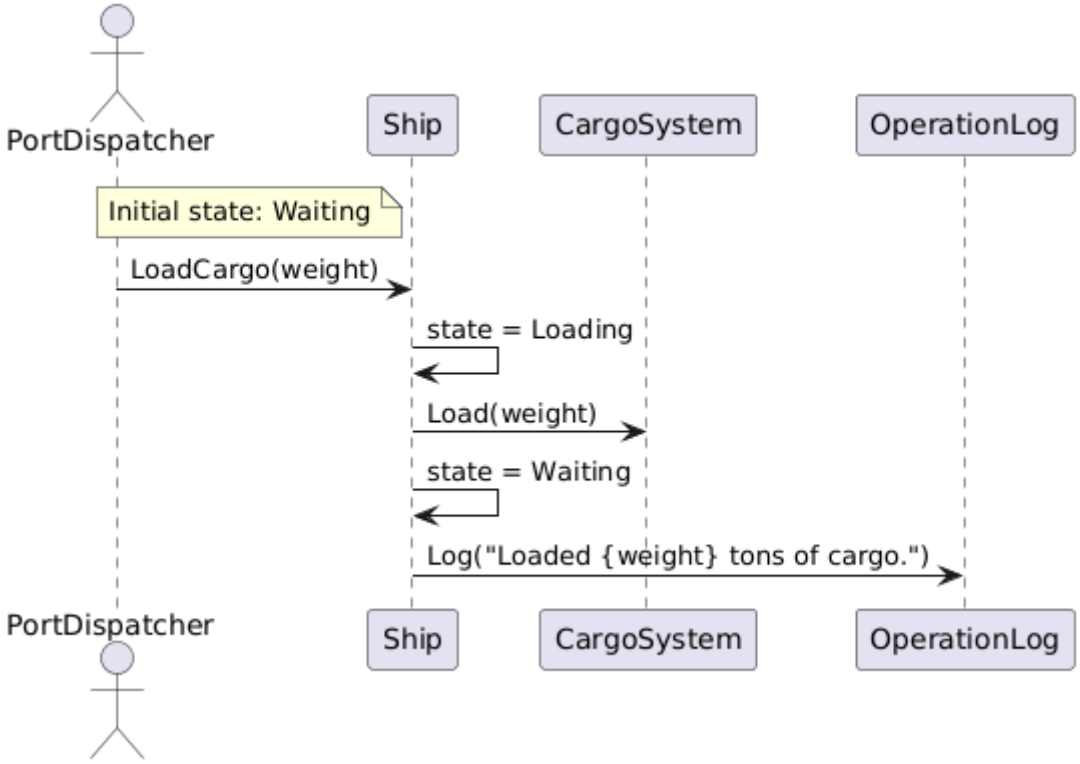

Рис. 6.5

Диаграмма последовательности «Выполнить погрузку»

Диаграмма последовательности варианта использования «Выполнить разгрузку» приведена на рисунке 6.6.

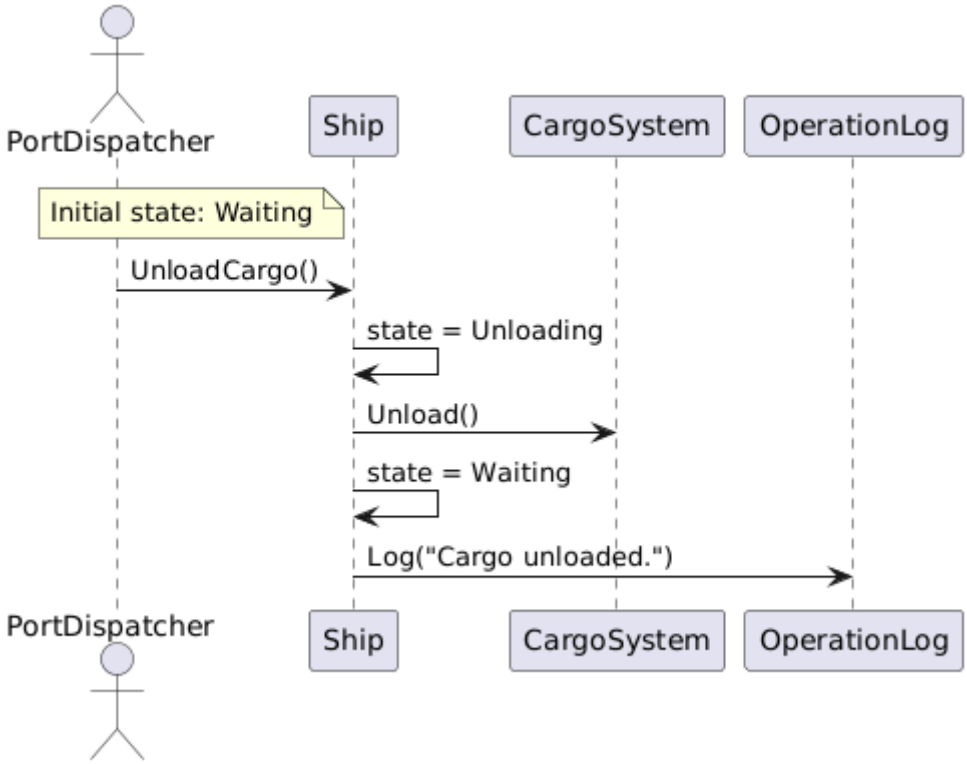

Рис. 6.6

Диаграмма последовательности «Выполнить разгрузку»

Диаграмма последовательности варианта использования «Мониторинг состояния» приведена на рисунке 6.7.

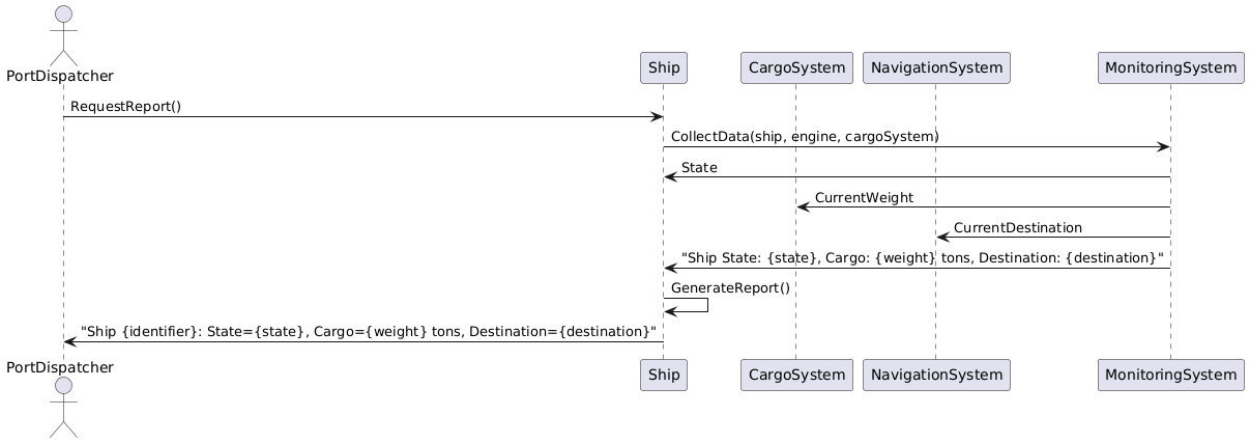

Рис. 6.7

Диаграмма последовательности «Мониторинг состояния»

Диаграмма последовательности варианта использования «Выполнить навигацию» приведена на рисунке 6.8.

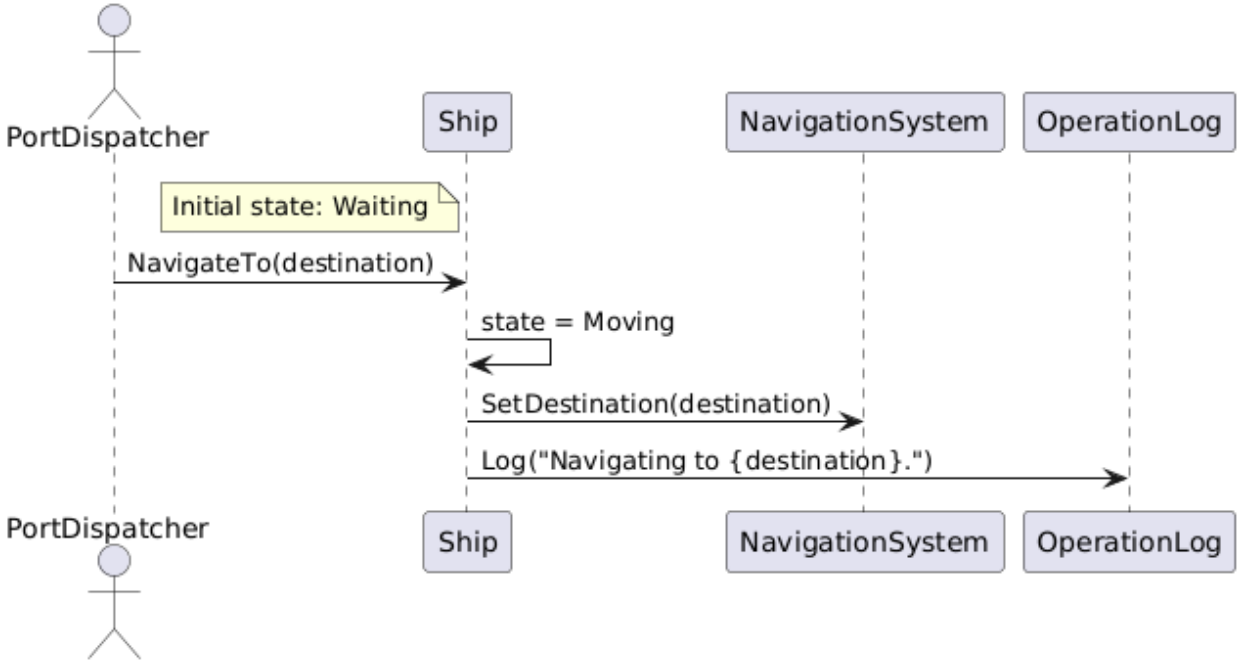

Рис. 6.8

Диаграмма последовательности «Выполнить навигацию»

```
[Fact]

public void Start_WhenStopped_ShouldTransitionToWaiting()

{

    _ship.Start();

    Assert.Equal(ShipState.Waiting, _ship.State);

    _engineMock.Verify(e => e.TurnOn(), Times.Once());

    _operationLogMock.Verify(l => l.Log("Ship started and moved to Waiting
state."), Times.Once());

}
```

```
[Fact]
public void Stop_WhenWaiting_ShouldTransitionToStopped()
{
    _ship.Start();
    _ship.Stop();
    Assert.Equal(ShipState.Stopped, _ship.State);
    _engineMock.Verify(e => e.TurnOff(), Times.Once());
    _operationLogMock.Verify(l => l.Log("Ship stopped."), Times.Once());
}
[Fact]
public void LoadCargoAndNavigate_ShouldTransitionThroughStates()
{
    _ship.Start();
    _cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);
    _ship.LoadCargo(100.0);
    Assert.Equal(ShipState.Waiting, _ship.State);
    _cargoSystemMock.Verify(c => c.Load(100.0), Times.Once());
    _operationLogMock.Verify(l => l.Log("Loaded 100 tons of cargo."),
Times.Once());


    _ship.NavigateTo("Port B");
    Assert.Equal(ShipState.Moving, _ship.State);
    _navigationSystemMock.Verify(n => n.SetDestination("Port B"),
Times.Once());
    _operationLogMock.Verify(l => l.Log("Navigating to Port B."),
Times.Once());
}
```

Контрольный пример

Приведём пример для следующей последовательности состояний: Ожидание → Погрузка → Ожидание → Движение → Ожидание. Состояния соответствующих объектов приведены на рисунках 6.9-6.12.

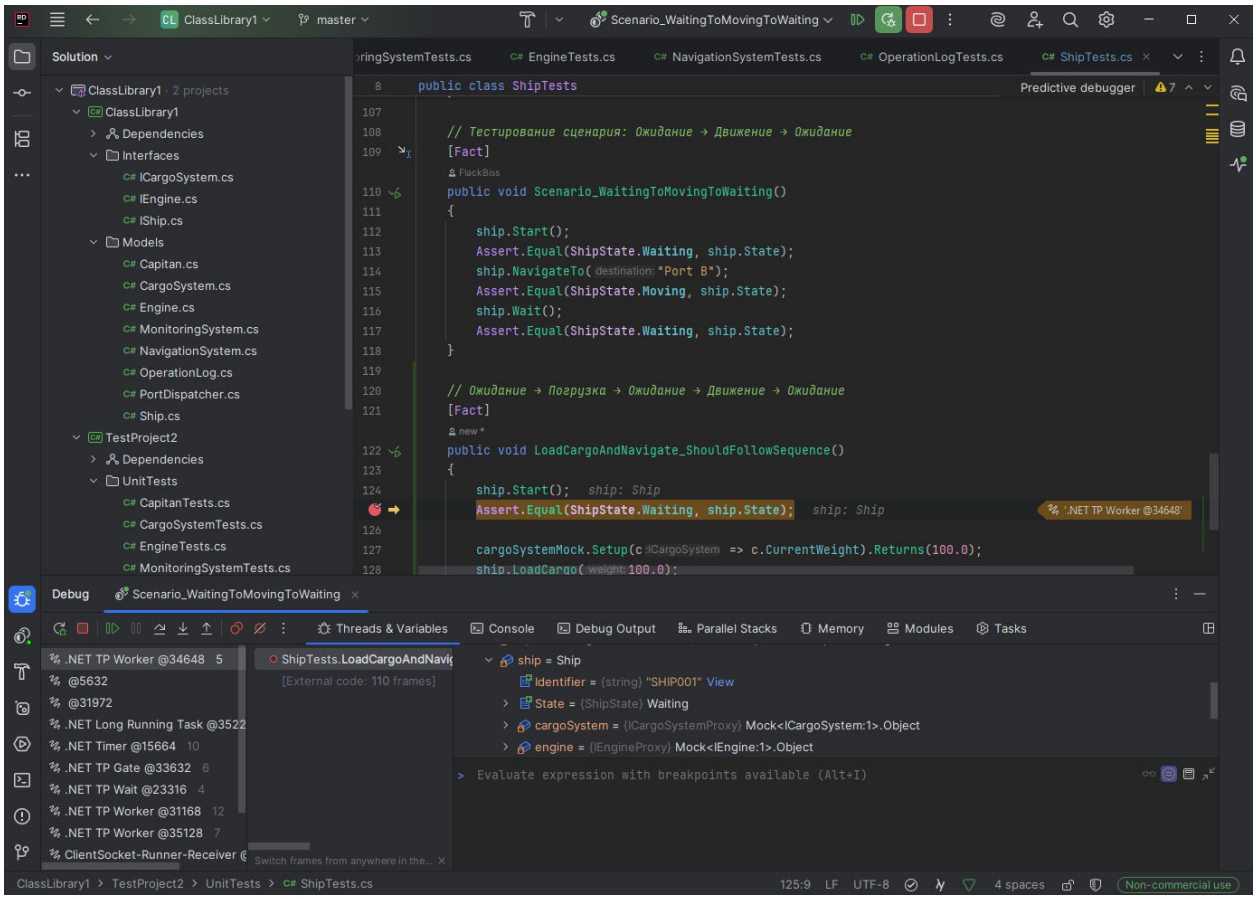

Рис. 6.9

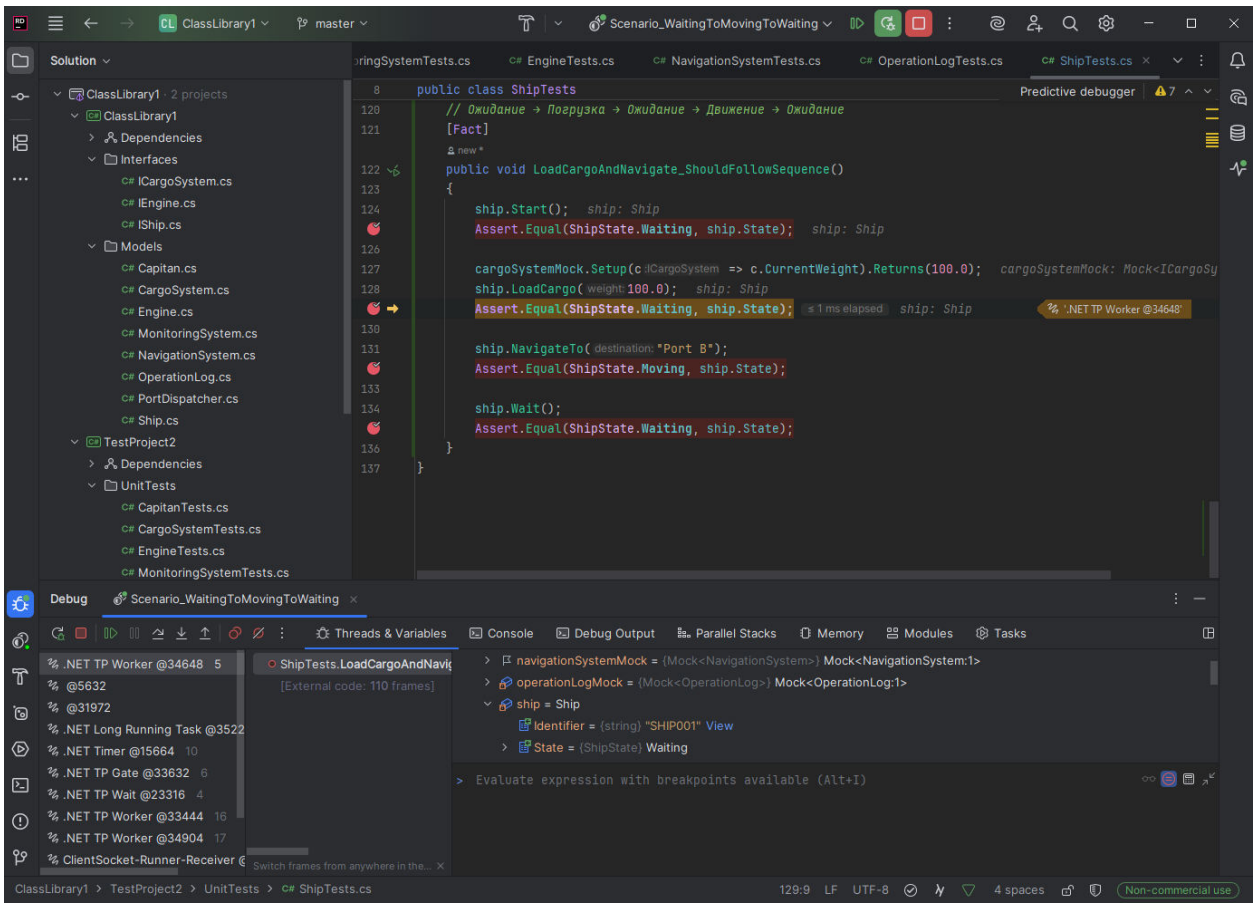
ClassLibrary1
master
Scenario_WaitingToMovingToWaiting
Solution
EngineTests.cs
NavigationSystemTests.cs
OperationLogTests.cs
ShipTests.cs
Predictive debugger
public class ShipTests
// Ожидание → Погрузка → Ожидание → Движение → Ожидание
[Fact]
public void LoadCargoAndNavigate_ShouldFollowSequence()
{
ship.Start(); ship: Ship
Assert.Equal(ShipState.Waiting, ship.State); ship: Ship
cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0); cargoSystemMock: Mock<ICargoSy
ship.LoadCargo(weight: 100.0); ship: Ship
Assert.Equal(ShipState.Waiting, ship.State); ≤ 1 ms elapsed ship: Ship
.NET TP Worker @34648
ship.NavigateTo(destination: "Port B");
Assert.Equal(ShipState.Moving, ship.State);
ship.Wait();
Assert.Equal(ShipState.Waiting, ship.State);
}
}
ClassLibrary1 · 2 projects
ClassLibrary1
Dependencies
Interfaces
ICargoSystem.cs
IEngine.cs
IShip.cs
Models
Capitan.cs
CargoSystem.cs
Engine.cs
MonitoringSystem.cs
NavigationSystem.cs
OperationLog.cs
PortDispatcher.cs
Ship.cs
TestProject2
Dependencies
UnitTests
CapitanTests.cs
CargoSystemTests.cs
EngineTests.cs
MonitoringSystemTests.cs
Debug
Scenario_WaitingToMovingToWaiting
Threads & Variables
Console
Debug Output
Parallel Stacks
Memory
Modules
Tasks
.NET TP Worker @34648 5
@5632
@31972
.NET Long Running Task @3522
.NET Timer @15664 10
.NET TP Gate @33632 6
.NET TP Wait @23316 4
.NET TP Worker @33444 16
.NET TP Worker @34904 17
ClientSocket-Runner-Receiver
ShipTests.LoadCargoAndNavig
[External code: 110 frames]
navigationSystemMock = {Mock<NavigationSystem>} Mock<NavigationSystem:1>
operationLogMock = {Mock<OperationLog>} Mock<OperationLog:1>
ship = Ship
Identifier = {string} "SHIP001" View
State = {ShipState} Waiting
Evaluate expression with breakpoints available (Alt+I)
ClassLibrary1 › TestProject2 › UnitTests › ShipTests.cs
129:9 LF UTF-8 4 spaces
Non-commercial use

Рис. 6.10

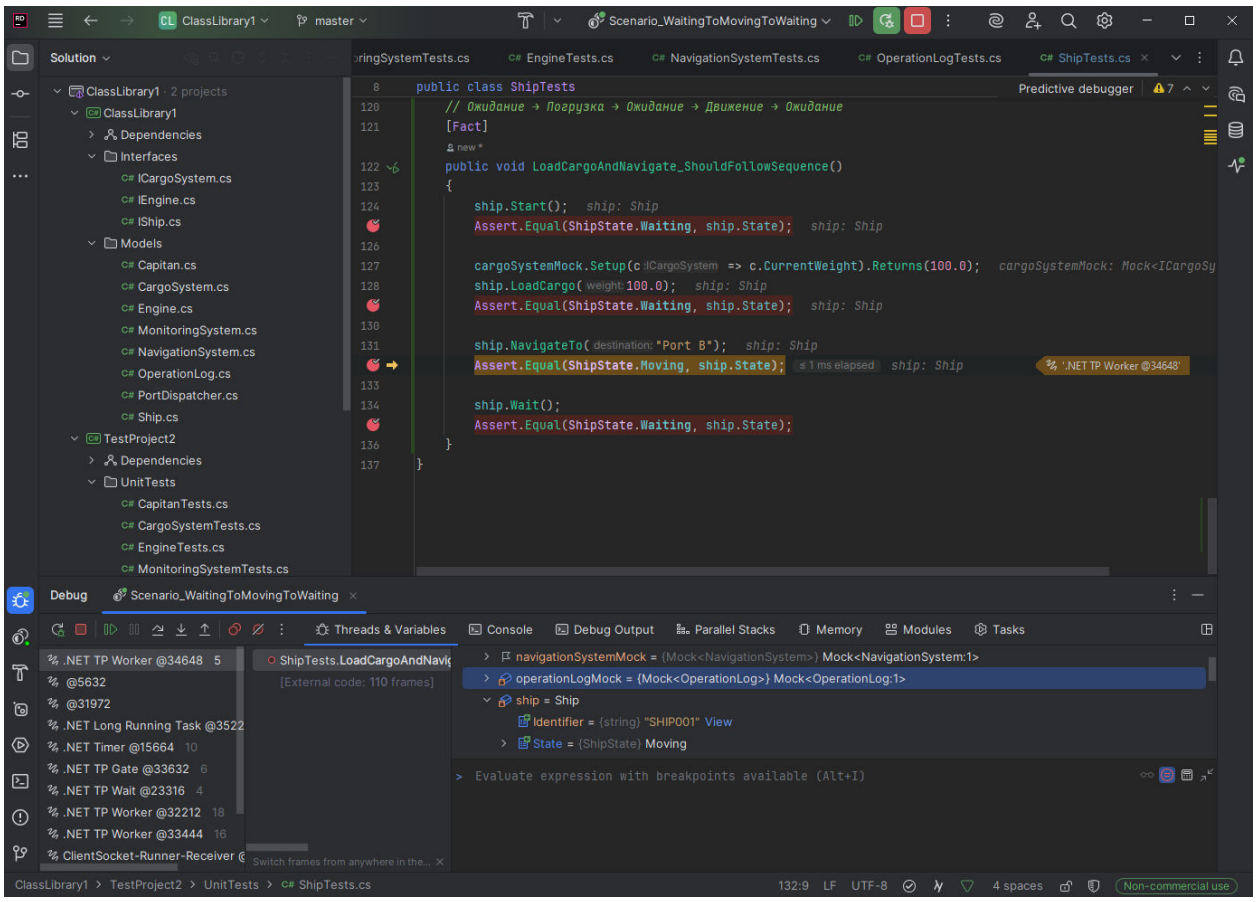
ClassLibrary1
master
Scenario_WaitingToMovingToWaiting
Solution
ClassLibrary1 · 2 projects
ClassLibrary1
Dependencies
Interfaces
ICargoSystem.cs
IEngine.cs
IShip.cs
Models
Capitan.cs
CargoSystem.cs
Engine.cs
MonitoringSystem.cs
NavigationSystem.cs
OperationLog.cs
PortDispatcher.cs
Ship.cs
TestProject2
Dependencies
UnitTests
CapitanTests.cs
CargoSystemTests.cs
EngineTests.cs
MonitoringSystemTests.cs
EngineTests.cs
NavigationSystemTests.cs
OperationLogTests.cs
ShipTests.cs
Predictive debugger
public class ShipTests
// Ожидание → Погрузка → Ожидание → Движение → Ожидание
[Fact]
new *
public void LoadCargoAndNavigate_ShouldFollowSequence()
{
ship.Start(); ship: Ship
Assert.Equal(ShipState.Waiting, ship.State); ship: Ship
cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0); cargoSystemMock: Mock<ICargoSy
ship.LoadCargo(weight: 100.0); ship: Ship
Assert.Equal(ShipState.Waiting, ship.State); ship: Ship
ship.NavigateTo(destination: "Port B"); ship: Ship
Assert.Equal(ShipState.Moving, ship.State); ≤ 1 ms elapsed ship: Ship
'.NET TP Worker @34648'
ship.Wait();
Assert.Equal(ShipState.Waiting, ship.State);
}
}
Debug
Scenario_WaitingToMovingToWaiting
Threads & Variables
Console
Debug Output
Parallel Stacks
Memory
Modules
Tasks
.NET TP Worker @34648 5
@5632
@31972
.NET Long Running Task @3522
.NET Timer @15664 10
.NET TP Gate @33632 6
.NET TP Wait @23316 4
.NET TP Worker @32212 18
.NET TP Worker @33444 16
ClientSocket-Runner-Receiver
ShipTests.LoadCargoAndNavig
[External code: 110 frames]
navigationSystemMock = {Mock<NavigationSystem>} Mock<NavigationSystem:1>
operationLogMock = {Mock<OperationLog>} Mock<OperationLog:1>
ship = Ship
Identifier = {string} "SHIP001" View
State = {ShipState} Moving
Evaluate expression with breakpoints available (Alt+I)
Switch frames from anywhere in the...
ClassLibrary1 › TestProject2 › UnitTests › ShipTests.cs
132:9 LF UTF-8 4 spaces
Non-commercial use

Рис. 6.11

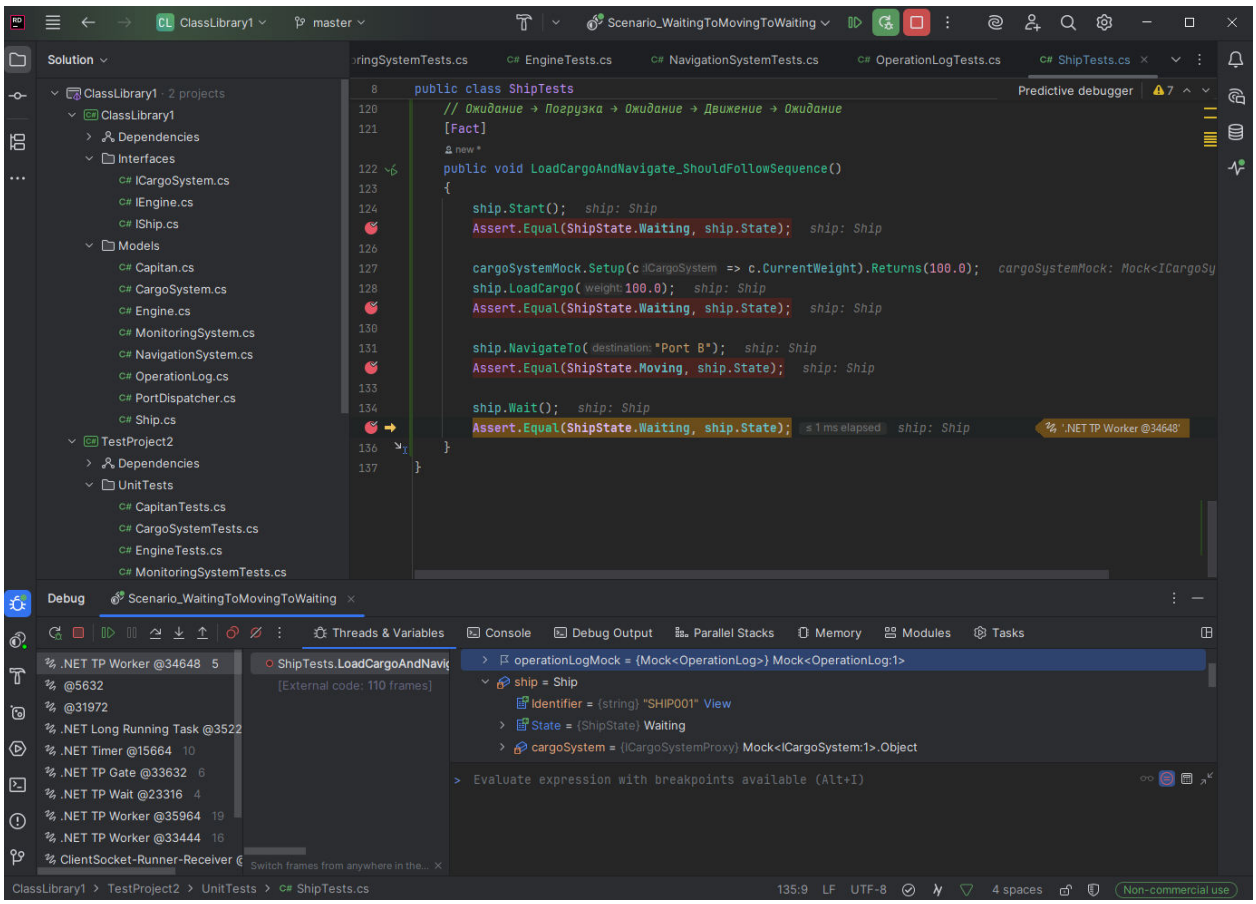

Рис. 6.12

```
// Ожидание → Погрузка → Ожидание → Движение → Ожидание
[Fact]
public void LoadCargoAndNavigate_ShouldFollowSequence()
{
    _ship.Start();
    Assert.Equal(ShipState.Waiting, _ship.State);

    _cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);
    _ship.LoadCargo(100.0);
    Assert.Equal(ShipState.Waiting, _ship.State);
    _operationLogMock.Verify(l => l.Log("Loaded 100 tons of cargo."),
Times.Once());
```

```
    _ship.NavigateTo("Port B");

    Assert.Equal(ShipState.Moving, _ship.State);

    _operationLogMock.Verify(l => l.Log("Navigating to Port B."),
Times.Once());



    _ship.Wait();

    Assert.Equal(ShipState.Waiting, _ship.State);

    _operationLogMock.Verify(l => l.Log("Ship Waiting."), Times.Once());

}
```

7. Текст программы

7.1. Объект «Ship»

```
using WeatherApp.Interfaces;

namespace WeatherApp.Models;

public class Ship : IShip
{
    private readonly string identifier; // Идентификатор судна
(константа)
    private ShipState state; // Текущее состояние судна
    private readonly IEngine engine; // Ссылка на двигатель
    private readonly ICargoSystem cargoSystem; // Ссылка на
грузовую систему
    private readonly NavigationSystem navigationSystem; //
Ссылка на навигационную систему
    private readonly OperationLog operationLog; // Ссылка на
журнал операций

    public Ship(string identifier, IEngine engine, ICargoSystem
cargoSystem, NavigationSystem navigationSystem,
        OperationLog operationLog)
    {
        this.identifier = identifier;
        this.engine = engine;
        this.cargoSystem = cargoSystem;
        this.navigationSystem = navigationSystem;
        this.operationLog = operationLog;
        this.state = ShipState.Stopped; // Начальное состояние —
Останов
    }

    public string Identifier => identifier;
    public virtual ShipState State => state;

    public void Start()
    {
        if (state == ShipState.Stopped)
        {
            engine.TurnOn();
            state = ShipState.Waiting;
            operationLog.Log("Ship started and moved to Waiting
state.");
        }
```

```
    }

    public void Stop()
    {
        if (state == ShipState.Moving || state ==
ShipState.Waiting)
        {
            engine.TurnOff();
            state = ShipState.Stopped;
            operationLog.Log("Ship stopped.");
        }
    }

    public void Wait()
    {
        if (state != ShipState.Waiting)
        {
            engine.TurnOn();
            state = ShipState.Waiting;
            operationLog.Log("Ship Waiting.");
        }
    }

    public void LoadCargo(double weight)
    {
        if (state == ShipState.Waiting)
        {
            state = ShipState.Loading;
            cargoSystem.Load(weight);
            state = ShipState.Waiting;
            operationLog.Log($"Loaded {weight} tons of cargo.");
        }
    }

    public void UnloadCargo()
    {
        if (state == ShipState.Waiting)
        {
            state = ShipState.Unloading;
            cargoSystem.Unload();
            state = ShipState.Waiting;
            operationLog.Log("Cargo unloaded.");
        }
    }

    public void NavigateTo(string destination)
```

```
    {
        if (state == ShipState.Waiting)
        {
            state = ShipState.Moving;
            navigationSystem.SetDestination(destination);
            operationLog.Log($"Navigating to {destination}.");
        }
    }

    public string GenerateReport()
    {
        return
            $"Ship {identifier}: State={state},
Cargo={cargoSystem.CurrentWeight} tons,
Destination={navigationSystem.CurrentDestination}";
    }
}
```

7.2. Объект «Captain»

```
namespace WeatherApp.Models;

public enum CaptainState
{
    Idle,        // Бездействует
    Commanding   // Командует
}

public class Captain
{
    private CaptainState state;

    public Captain()
    {
        this.state = CaptainState.Idle;
    }

    public CaptainState State => state;

    public void GiveCommand(string command)
    {
        state = CaptainState.Commanding;
        state = CaptainState.Idle;
```

```
    }
}
```

7.3. Объект «CargoSystem»

```
using WeatherApp.Interfaces;

namespace WeatherApp.Models;

public enum CargoState
{
    Empty, // Пустая
    Loading, // Загружается
    Full // Полная
}

public class CargoSystem : ICargoSystem
{
    private double currentWeight; // Текущий вес груза
    private readonly double maxCapacity; // Максимальная
вместимость
    private CargoState state; // Состояние грузовой системы

    public CargoSystem(double maxCapacity)
    {
        this.maxCapacity = maxCapacity;
        this.currentWeight = 0;
        this.state = CargoState.Empty;
    }

    public double CurrentWeight => currentWeight;
    public double MaxCapacity => maxCapacity;
    public CargoState State => state;

    public void Load(double weight)
    {
        if (currentWeight + weight <= maxCapacity)
        {
            state = CargoState.Loading;
            currentWeight += weight;
            state = currentWeight >= maxCapacity ?
CargoState.Full : CargoState.Empty;
        }
        else
```

```
        {
            throw new InvalidOperationException("Maximum weight
cannot be more than maximum capacity");
        }
    }

    public void Unload()
    {
        currentWeight = 0;
        state = CargoState.Empty;
    }
}
```

7.4. Объект «Engine»

```
using WeatherApp.Interfaces;

namespace WeatherApp.Models;

public enum EngineState
{
    Off, // Выключен
    On, // Работает
    Faulty // Неисправен
}

public class Engine : IEngine
{
    private double power; // Мощность двигателя
    private EngineState state; // Состояние двигателя

    public Engine(double power)
    {
        this.power = power;
        this.state = EngineState.Off; // Начальное состояние —
Выключен
    }

    public double Power => power;
    public EngineState State => state;

    public void TurnOn()
    {
        if (state == EngineState.Off)
        {
            state = EngineState.On;
        }
```

```
    }

    public void TurnOff()
    {
        if (state == EngineState.On)
        {
            state = EngineState.Off;
        }
    }

    public void CheckState()
    {
        if (power < 100)
        {
            state = EngineState.Faulty;
        }
    }
}
```

7.5. Объект «MonitoringSystem»

```
using WeatherApp.Interfaces;

namespace WeatherApp.Models;

public class MonitoringSystem
{
    private bool isMonitoring; // Состояние мониторинга

    public MonitoringSystem()
    {
        this.isMonitoring = false; // Начальное состояние — Неактивна
    }

    public bool IsMonitoring => isMonitoring;

    public void StartMonitoring()
    {
```

```
        this.isMonitoring = true;
    }

    public void StopMonitoring()
    {
        this.isMonitoring = false;
    }

    public string CollectData(IShip ship, IEngine engine,
ICargoSystem cargoSystem)
    {
        if (isMonitoring)
        {
            return $"Ship State: {ship.State}, Engine State:
{engine.State}, Cargo: {cargoSystem.CurrentWeight} tons";
        }
        return "Monitoring is not active.";
    }
}
```

7.6. Объект «NavigationSystem»

```
namespace WeatherApp.Models;

public class NavigationSystem
{
    private string currentDestination;  // Текущий пункт
назначения
    private bool isActive;              // Состояние активности
системы

    public NavigationSystem()
    {
        this.currentDestination = "None";
        this.isActive = false; // Начальное состояние —
Неактивна
    }
```

```
    public virtual string CurrentDestination =>
currentDestination;
    public virtual bool IsActive => isActive;

    public virtual void SetDestination(string destination)
    {
        this.currentDestination = destination;
        this.isActive = true;
    }

    public void StopNavigation()
    {
        this.currentDestination = "None";
        this.isActive = false;
    }
}
```

7.7. Объект «OperationLog»

```
namespace WeatherApp.Models;

public class OperationLog
{
    private List<string> logs; // Список логов

    public OperationLog()
    {
        this.logs = new List<string>();
    }

    public IReadOnlyList<string> Logs => logs.AsReadOnly();

    public virtual void Log(string message)
    {
        logs.Add($"{DateTime.Now}: {message}");
    }

    public void Clear()
```

```
    {
        logs.Clear();
    }
}
```

7.8. Тест «PortDispatcher»

```
using WeatherApp.Interfaces;

namespace WeatherApp.Models;

public class PortDispatcher
{
    public string RequestReport(IShip ship)
    {
        return ship.GenerateReport();
    }

    public void SendCommand(IShip ship, string command)
    {
        switch (command)
        {
            case "start":
                ship.Start();
                break;
            case "stop":
                ship.Stop();
                break;
            case "load":
                ship.LoadCargo(100); // Пример веса
                break;
            case "unload":
                ship.UnloadCargo();
                break;
            case "navigate":
                ship.NavigateTo("Port B");
                break;
        }
    }
}
```

7.9. Тест «CaptainTests»

```
using WeatherApp.Models;

namespace WeatherApp.Test.UnitTests;

public class CaptainTests
{
    private readonly Captain captain;

    public CaptainTests()
    {
        captain = new Captain();
    }

    // Тестирование свойства
    [Fact]
    public void State_InitiallyIdle()
    {
        Assert.Equal(CaptainState.Idle, captain.State);
    }

    // Тестирование методов и состояний
    [Fact]
    public void GiveCommand_ChangesStateToCommandingThenIdle()
    {
        captain.GiveCommand("navigate");
        Assert.Equal(CaptainState.Idle, captain.State); // После
выполнения команды возвращается в Idle
    }
}
```

7.10. Тест «CargoSystemTests»

```
using WeatherApp.Models;

namespace WeatherApp.Test.UnitTests;

public class CargoSystemTests
{
    private readonly CargoSystem _cargoSystem;

    public CargoSystemTests()
```

```
    {
        _cargoSystem = new CargoSystem(1000.0);
    }

    // Тест атрибута MaxCapacity
    [Fact]
    public void MaxCapacity_ShouldReturnCorrectValue()
    {
        Assert.Equal(1000.0, _cargoSystem.MaxCapacity);
    }

    // Тест метода Load
    [Fact]
    public void Load_ShouldUpdateWeightAndState()
    {
        _cargoSystem.Load(500.0);
        Assert.Equal(500.0, _cargoSystem.CurrentWeight);
        Assert.Equal(CargoState.Empty, _cargoSystem.State);
    }

    // Тест метода Load при достижении максимальной вместимости
    [Fact]
    public void Load_AtMaxCapacity_ShouldSetStateToFull()
    {
        _cargoSystem.Load(1000.0);
        Assert.Equal(1000.0, _cargoSystem.CurrentWeight);
        Assert.Equal(CargoState.Full, _cargoSystem.State);
    }

    // Тест метода Unload
    [Fact]
    public void Unload_ShouldResetWeightAndState()
    {
        _cargoSystem.Load(500.0);
        _cargoSystem.Unload();
        Assert.Equal(0.0, _cargoSystem.CurrentWeight);
        Assert.Equal(CargoState.Empty, _cargoSystem.State);
    }

    // Тест исключения для Load при превышении вместимости
    [Fact]
    public void Load_WhenExceedsCapacity_ShouldThrowException()
    {
        Assert.Throws<InvalidOperationException>(() =>
_cargoSystem.Load(1100.0));
```

```
        }
}
```