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 4](#_Toc196662231)

[1.1. Цель работы 4](#_Toc196662232)

[1.2. Описание предметной области 4](#_Toc196662233)

[1.3. Постановка задачи 5](#_Toc196662234)

[1.4. Требования к системе 5](#_Toc196662235)

[2. Описание предметной области 7](#_Toc196662236)

[3. Описание системы 10](#_Toc196662237)

[3.1. Объект «Портовый диспетчер» 10](#_Toc196662238)

[3.2. Объект «Судно» 11](#_Toc196662239)

[3.3. Объект «Капитан» 12](#_Toc196662240)

[3.4. Объект «Двигатель» 13](#_Toc196662241)

[3.5. Объект «Грузовая система» 15](#_Toc196662242)

[3.6. Объект «Навигационная система» 16](#_Toc196662243)

[3.7. Объект «Система мониторинга» 17](#_Toc196662244)

[3.8. Объект «Журнал операций» 18](#_Toc196662245)

[3.9. Схема системы 19](#_Toc196662246)

[3.10. Описание взаимодействия между объектами 20](#_Toc196662247)

[4. Тестирование свойств и методов объектов 22](#_Toc196662248)

[4.1. Тестирование объекта «Ship» 22](#_Toc196662249)

[4.2. Тестирование объекта «CargoSystem» 24](#_Toc196662250)

[4.3. Тестирование объекта «Engine» 25](#_Toc196662251)

[4.4. Тестирование объекта «NavigationSystem» 27](#_Toc196662252)

[4.5. Тестирование объекта «Capitan» 28](#_Toc196662253)

[4.6. Тестирование объекта «MonitoringSystem» 28](#_Toc196662254)

[4.7. Тестирование объекта «OperationLog» 29](#_Toc196662255)

[4.8. Тестирование объекта «PortDispatcher» 30](#_Toc196662256)

[5.1. Модель состояний системы 32](#_Toc196662257)

[5.2. Описание модели состояний системы 32](#_Toc196662258)

[5.3. Описание последовательностей состояний для тестирования 33](#_Toc196662259)

[6. Тестирование сценариев и вариантов использования 36](#_Toc196662260)

[6.1. Диаграмма вариантов использования 36](#_Toc196662261)

[Приведём диаграмму вариантов использования на рисунке 6.1. 36](#_Toc196662262)

[Варианты использования 36](#_Toc196662263)

[6.2. Диаграммы последовательности 38](#_Toc196662264)

[Контрольный пример 44](#_Toc196662265)

[7. Текст программы 50](#_Toc196662266)

[7.1. Объект «Ship» 50](#_Toc196662267)

[7.2. Объект «Captain» 52](#_Toc196662268)

[7.3. Объект «CargoSystem» 53](#_Toc196662269)

[7.4. Объект «Engine» 54](#_Toc196662270)

[7.5. Объект «MonitoringSystem» 55](#_Toc196662271)

[7.6. Объект «NavigationSystem» 56](#_Toc196662272)

[7.7. Объект «OperationLog» 57](#_Toc196662273)

[7.8. Тест «PortDispatcher» 58](#_Toc196662274)

[7.9. Тест «CaptainTests» 59](#_Toc196662275)

[7.10. Тест «CargoSystemTests» 59](#_Toc196662276)

[7.11 Тест «EngineTests.cs» 61](#_Toc196662277)

[7.12 Тест «NavigationSystemTests.cs» 62](#_Toc196662278)

[7.13 Тест «PortDispatcher.cs» 62](#_Toc196662279)

[7.14 Тест «ShipTests.cs» 67](#_Toc196662280)

1. Задание
   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. 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. 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. Требования к системе
7. Система должна корректно переходить между состояниями (Останов, Ожидание, Движение, Погрузка, Разгрузка) при вызове соответствующих методов, таких как Start(), Stop(), LoadCargo(), NavigateTo().
8. Объекты системы (Ship, CargoSystem, Engine, NavigationSystem и др.) должны взаимодействовать через четко определенные методы, например TurnOn(), SetDestination(), GenerateReport().
9. Все сценарии использования должны быть покрыты тестами, включая базовые (запуск/остановка) и комбинированные (погрузка с последующей навигацией).
10. Код должен быть размещен в репозитории GitHub с настроенными автоматическими тестами через GitHub Actions.
11. Документация должна включать UML-диаграммы и подробное описание всех процессов.
12. 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.

1. Описание системы
   1. Объект «Портовый диспетчер»

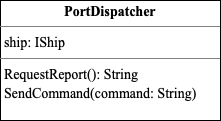


Рис. 3.1.

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.
  1. Объект «Судно»

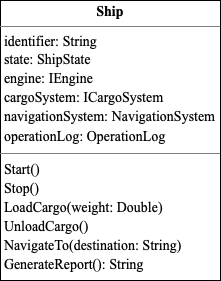


Рис. 3.2.

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».
  1. Объект «Капитан»

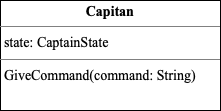
****

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.
  1. Объект «Двигатель»

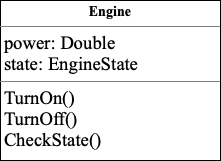
****

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».
  1. Объект «Грузовая система»

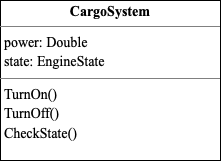
****

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».
  1. Объект «Навигационная система»

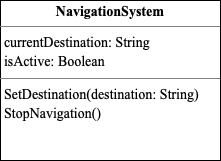
****

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».
  1. Объект «Система мониторинга»

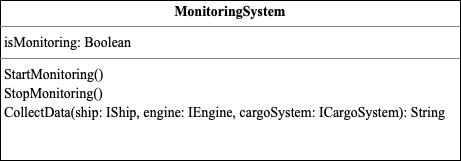


Рис. 3.7.

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».
  1. 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».
  1. Схема системы

Схема системы представлена на рисунке 3.9.

Схема системы «Управление судном»

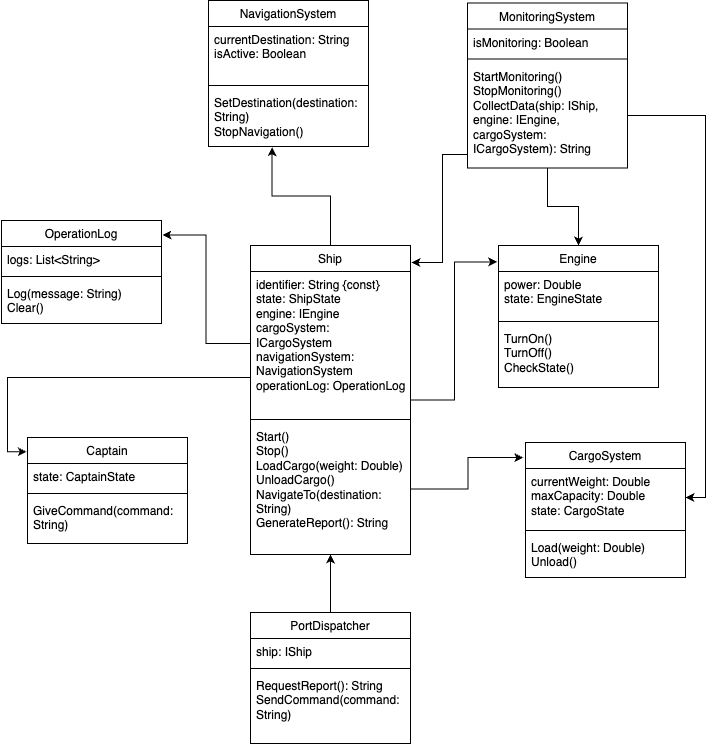


Рис. 3.9.

* 1. 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).

1. **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.

1. **Капитан:**

* Отдаёт команды через GiveCommand(), которые могут быть переданы судну (например, для навигации).
* Взаимодействует с судном (в перспективе через интеграцию).

1. **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().

1. **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().

1. **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().

1. **Система мониторинга:**

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

1. **Журнал операций:**

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

1. Тестирование свойств и методов объектов
   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);  
 }

* 1. 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));  
 }  
}

* 1. 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);  
 }  
}

* 1. 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);  
 }  
}

* 1. 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  
 }  
}

* 1. 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);  
 }  
}

* 1. 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);  
 }  
}

* 1. 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());  
 }  
}

1. **Тестирование состояний объектов**
   1. Модель состояний системы

Приведём схему модели состояний на рисунке 5.1.

Модель состояний системы



Рис. 5.1

* 1. 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.
   1. Описание последовательностей состояний для тестирования

Опишем последовательности состояний для тестирования системы управления судном:

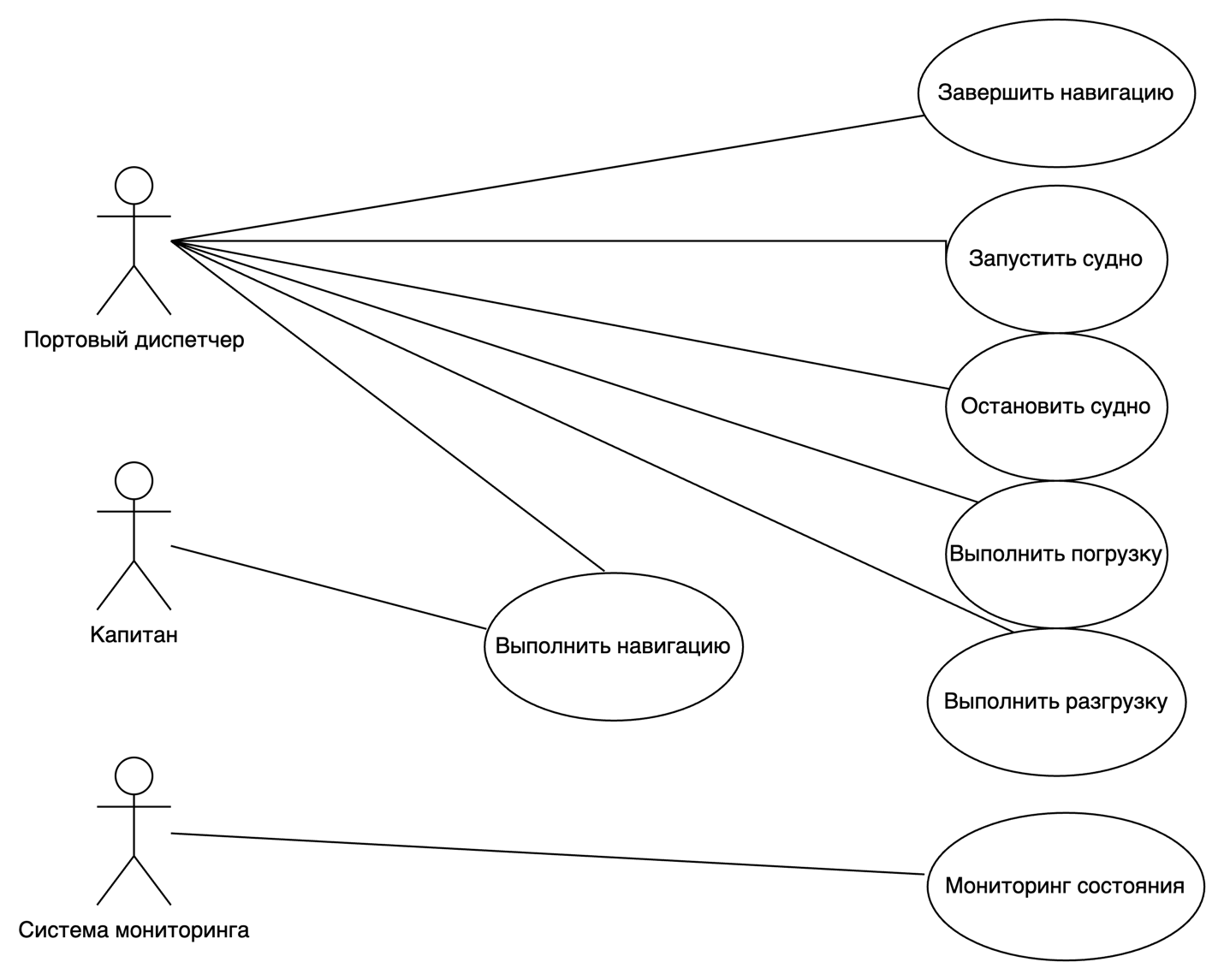
1. **Останов → Ожидание → Останов.**  
   Проверяет базовый цикл запуска и остановки системы. Тест должен убедиться, что система корректно переходит из Останов в Ожидание при вызове Start() и возвращается в Останов при вызове Stop().
2. **Ожидание → Погрузка → Ожидание.**  
   Проверяет процесс погрузки 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().
8. **Останов → Ожидание**  
   Проверяет базовый переход из состояния остановки в состояние ожидания. Тест должен убедиться, что система корректно переходит из Останов в Ожидание при вызове Start().
9. **Ожидание → Останов**  
   Проверяет базовый переход из состояния ожидания в состояние остановки. Тест должен убедиться, что система корректно переходит из Ожидание в Останов при вызове Stop().
10. **Ожидание → Погрузка → Ожидание**  
    Проверяет процесс погрузки грузов. Тест должен убедиться, что система корректно проходит через LoadCargo() из состояния Ожидание, переходит в Погрузка и возвращается в Ожидание.
11. **Ожидание → Движение**  
    Проверяет процесс начала навигации. Тест должен убедиться, что система переходит из Ожидание в Движение при вызове NavigateTo().
12. **Движение → Ожидание**  
    Проверяет завершение навигации. Тест должен убедиться, что система переходит из Движение в Ожидание при вызове Wait().
13. **Останов → Ожидание (через Wait)**  
    Проверяет альтернативный переход в состояние ожидания из состояния остановки. Тест должен убедиться, что система корректно переходит из Останов в Ожидание при вызове Wait().
14. Ожидание → Движение → Останов

Проверяет сценарий навигации с последующей остановкой системы. Тест должен убедиться, что система переходит из Ожидание в Движение при вызове NavigateTo(), а затем переходит в Останов через Stop().

1. Тестирование сценариев и вариантов использования
   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».

* 1. Диаграммы последовательности

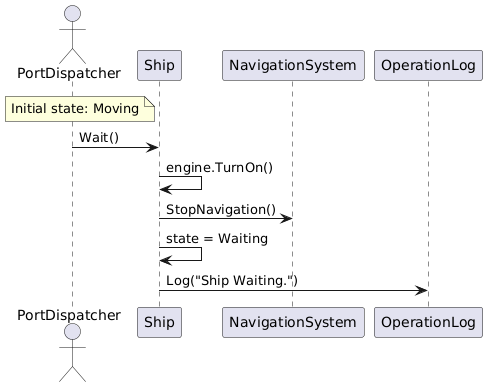
Диаграмма последовательности варианта использования «Завершить навигацию» приведена на рисунке 6.2.

Рис. 6.2

Диаграмма последовательности «Завершить навигацию»

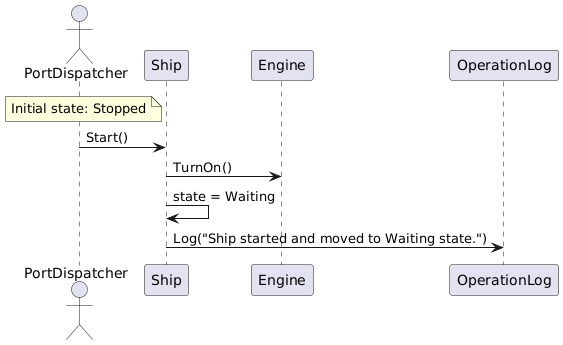
Диаграмма последовательности варианта использования «Запустить судно» приведена на рисунке 6.3.

Рис. 6.3

Диаграмма последовательности «Запустить судно»

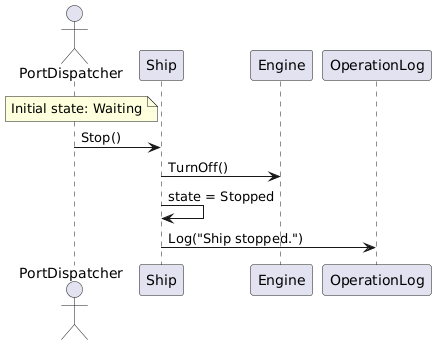
Диаграмма последовательности варианта использования «Остановить судно» приведена на рисунке 6.4.

Рис. 6.4

Диаграмма последовательности «Остановить судно»

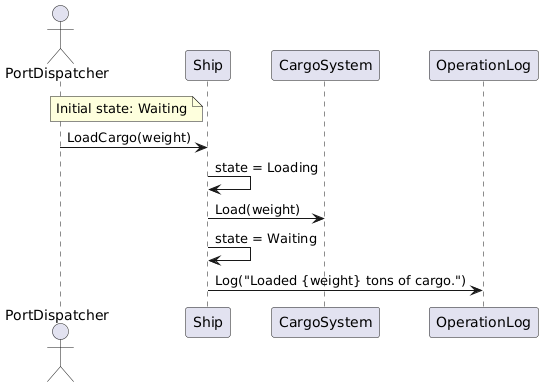
Диаграмма последовательности варианта использования «Выполнить погрузку» приведена на рисунке 6.5.

Рис. 6.5

Диаграмма последовательности «Выполнить погрузку»

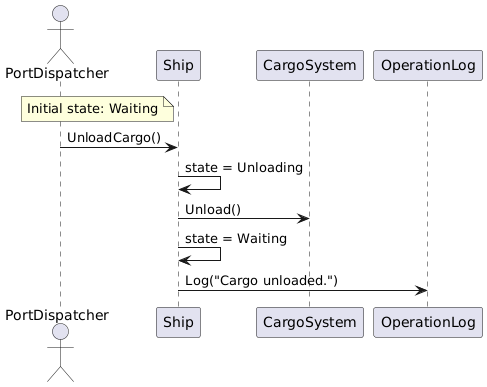
Диаграмма последовательности варианта использования «Выполнить разгрузку» приведена на рисунке 6.6.

Рис. 6.6

Диаграмма последовательности «Выполнить разгрузку»

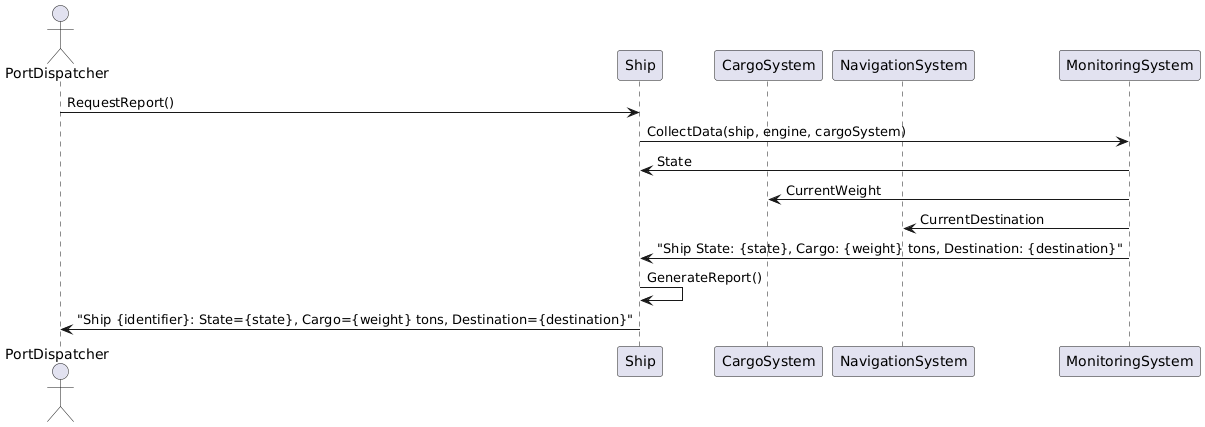
Диаграмма последовательности варианта использования «Мониторинг состояния» приведена на рисунке 6.7.

Рис. 6.7

Диаграмма последовательности «Мониторинг состояния»

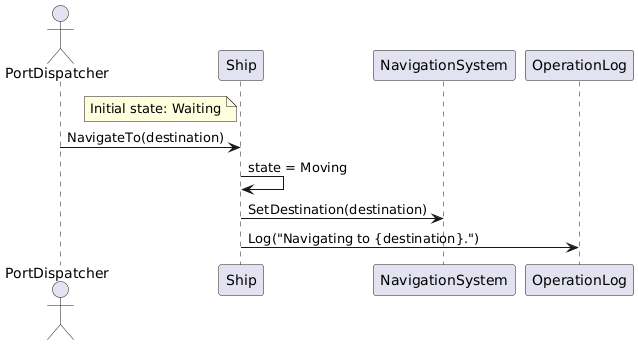
Диаграмма последовательности варианта использования «Выполнить навигацию» приведена на рисунке 6.8.

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: 7. Ожидание → Погрузка → Ожидание → Разгрузка → Ожидание → Движение → Ожидание. Состояния соответствующих объектов приведены на рисунках 6.9-6.12.

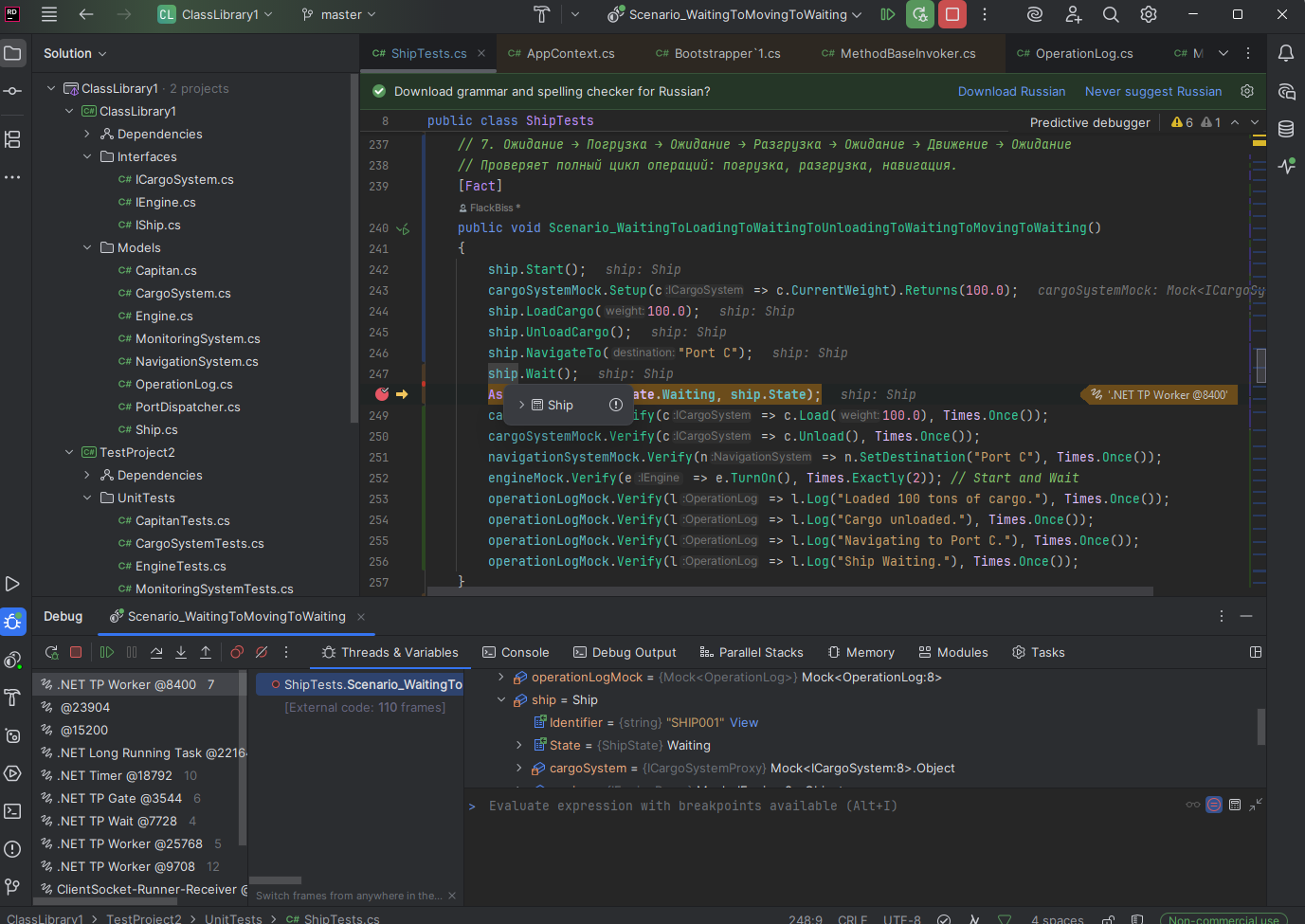


Рис. 6.9

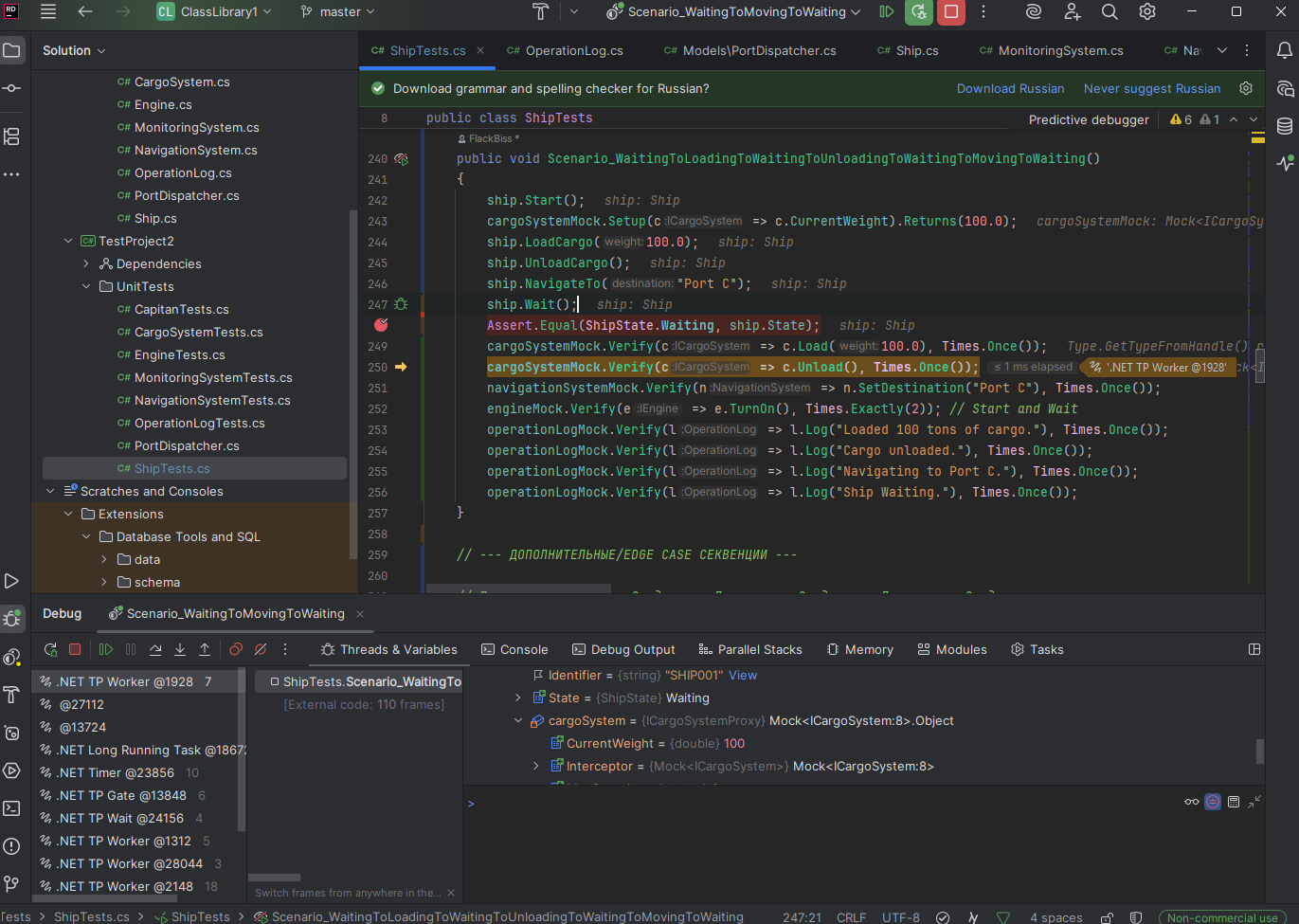


Рис. 6.10

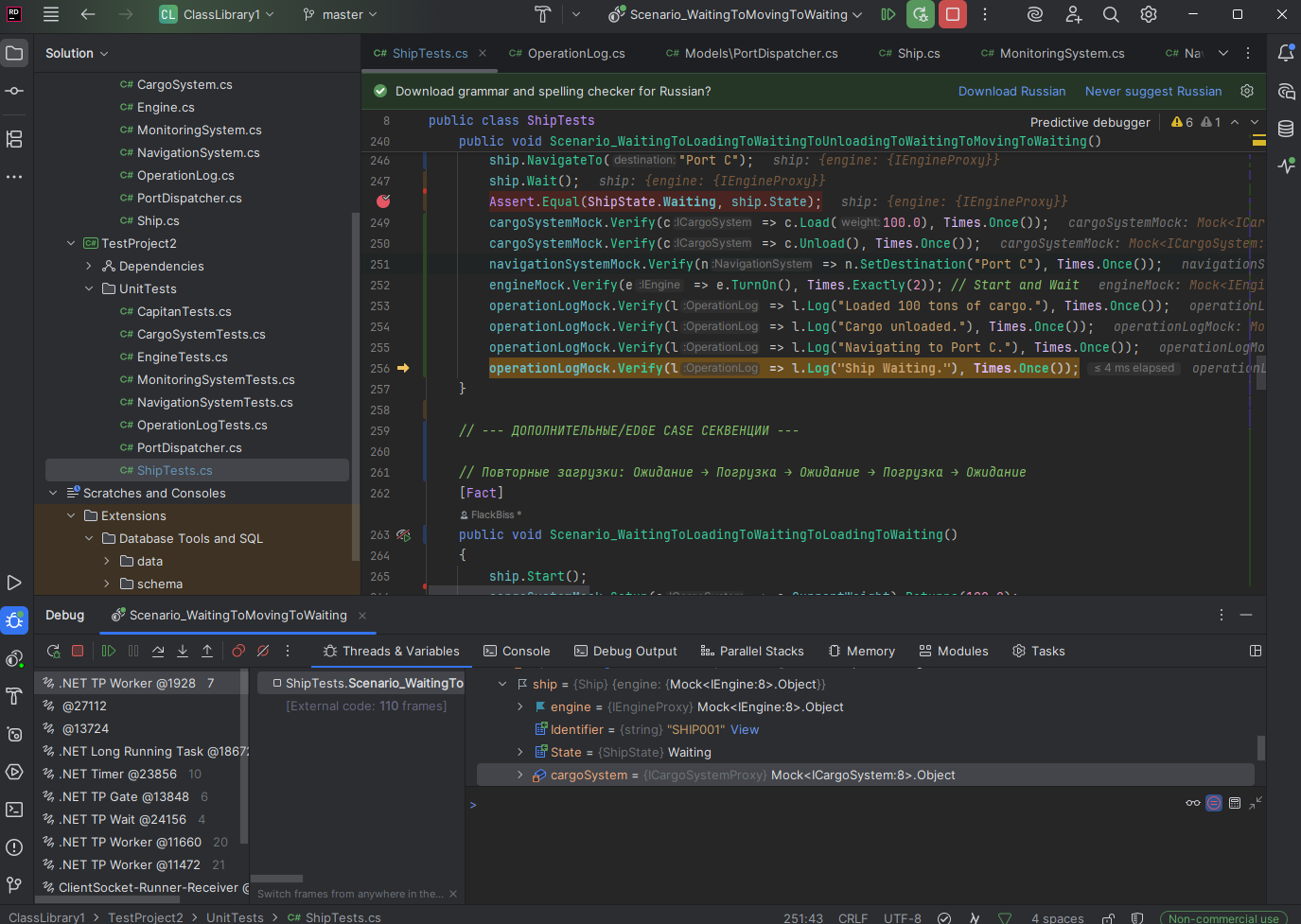


Рис. 6.11

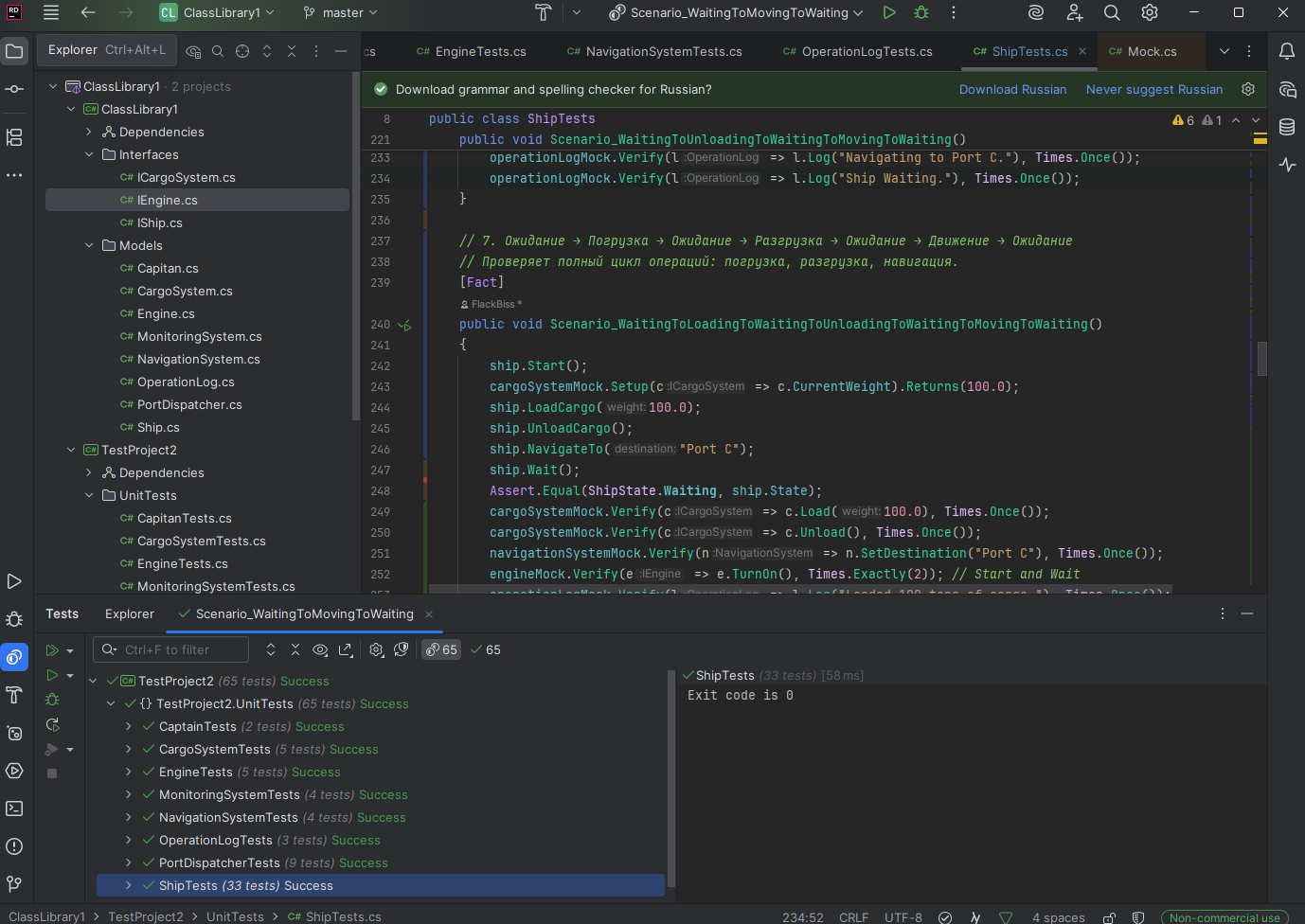


Рис. 6.12

Сценарии использования:

1. Запуск судна и получение статуса

**Участвуют:** PortDispatcher, Ship, Engine, OperationLog

1. **PortDispatcher** отправляет команду start судну (Ship).
2. **Ship** включает двигатель (Engine.TurnOn()), изменяет состояние на Waiting.
3. **Ship** записывает событие в журнал операций (OperationLog.Log(...)).
4. **PortDispatcher** запрашивает у судна статус (GenerateReport()).
5. **Ship** возвращает отчёт о состоянии.
6. **PortDispatcher** получает и отображает результат.
7. Погрузка груза

**Участвуют:** PortDispatcher, Ship, CargoSystem, OperationLog

1. **PortDispatcher** отправляет команду load с весом, например 200.
2. **Ship** вызывает LoadCargo(200); передаёт вызов в **CargoSystem.Load(200)**.
3. **CargoSystem** обновляет вес груза.
4. **Ship** пишет запись в журнал (OperationLog.Log(...)).
5. **PortDispatcher** запрашивает отчёт у судна.
6. **Ship** возвращает отчёт о текущем весе груза.
7. **PortDispatcher** получает результат.
8. Навигация

**Участвуют:** PortDispatcher, Ship, NavigationSystem, OperationLog

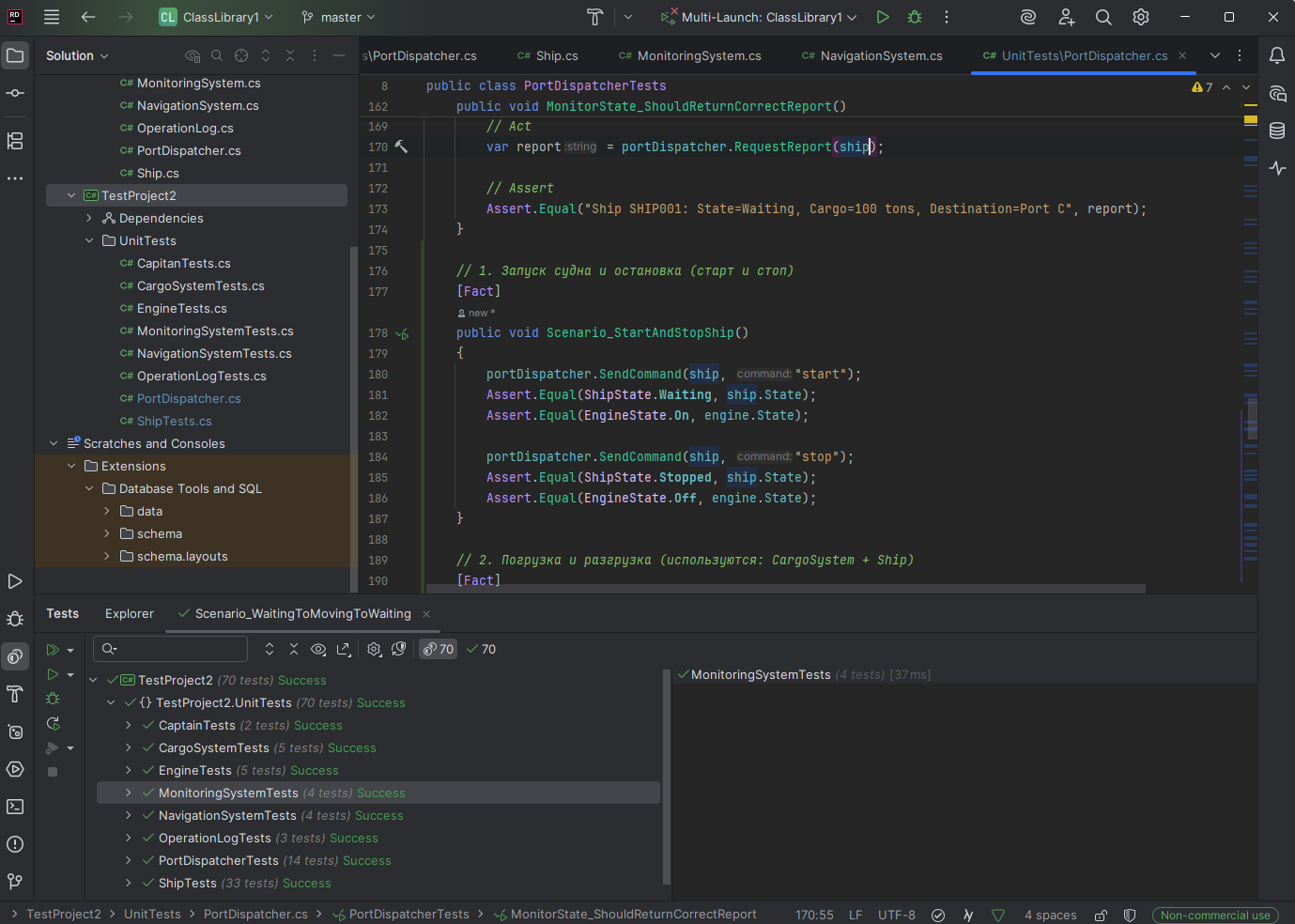
1. **PortDispatcher** отправляет команду navigate с пунктом “Port A”.
2. **Ship** вызывает NavigateTo("Port A"), передаёт вызов **NavigationSystem.SetDestination("Port A")**.
3. **Ship** пишет запись в журнал (OperationLog.Log(...)).
4. **PortDispatcher** запрашивает отчёт у судна.
5. **Ship** возвращает статус и текущий пункт назначения.
6. **PortDispatcher** получает результат.
7. Разгрузка

**Участвуют:** PortDispatcher, Ship, CargoSystem, OperationLog

1. **PortDispatcher** отправляет команду unload.
2. **Ship** вызывает UnloadCargo(), передаёт команду **CargoSystem.Unload()**.
3. **CargoSystem** обнуляет вес груза.
4. **Ship** пишет запись в журнал (OperationLog.Log(...)).
5. **PortDispatcher** запрашивает отчёт у судна.
6. **Ship** возвращает отчёт с информацией о грузе.
7. **PortDispatcher** получает результат.
8. Полный цикл эксплуатации (используются все объекты!)

**Участвуют:** PortDispatcher, Ship, Captain, Engine, CargoSystem, NavigationSystem, MonitoringSystem, OperationLog

1. **PortDispatcher** отправляет команду start.
2. **Ship** включает **Engine** и фиксирует запуск через **OperationLog**.
3. **PortDispatcher** отправляет команду load (например, 300).
4. **Ship** вызывает **CargoSystem.Load(300)** и логирует событие.
5. **PortDispatcher** отправляет команду navigate (“Port B”).
6. **Ship** вызывает **NavigationSystem.SetDestination("Port B")** и логирует навигацию.
7. **PortDispatcher** активирует мониторинг (MonitoringSystem.StartMonitoring()).
8. **MonitoringSystem** собирает данные с **Ship**, **Engine**, **CargoSystem**.
9. **PortDispatcher** отправляет команду unload.
10. **Ship** вызывает **CargoSystem.Unload()**, фиксирует действие в **OperationLog**.
11. **PortDispatcher** отправляет команду stop.
12. **Ship** выключает **Engine**, пишет в **OperationLog**.
13. **PortDispatcher** запрашивает итоговый отчёт у **Ship**.
14. **Ship** возвращает отчёт о финальном состоянии, который содержит данные о всех выполненных действиях.
15. **PortDispatcher** получает полный результат и отображает его.

**

*Рис. 6.13*

1. Текст программы
   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}";  
 }  
}

* 1. 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;  
 }  
}

* 1. 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;  
 }  
}

* 1. 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;  
 }  
 }  
}

* 1. 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.";  
 }  
}

* 1. 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;  
 }  
}

* 1. 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();  
 }  
}

* 1. 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;  
 }  
 }  
}

* 1. Тест «CaptainTests»

using WeatherApp.Models;  
  
namespace TestProject2.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  
 }  
}

* 1. 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));  
 }  
}

* 1. Тест «EngineTests.cs»

1. using ClassLibrary1.Models;  
     
   namespace TestProject2.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);  
    }  
   }
   1. Тест «NavigationSystemTests.cs»
2. using ClassLibrary1.Models;  
     
   namespace TestProject2.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);  
    }  
   }
   1. Тест «PortDispatcher.cs»
3. using ClassLibrary1.Interfaces;  
   using ClassLibrary1.Models;  
   using WeatherApp.Interfaces;  
   using Xunit;  
     
   namespace TestProject2.UnitTests;  
     
   public class PortDispatcherTests  
   {  
    private readonly IEngine engine;  
    private readonly ICargoSystem cargoSystem;  
    private readonly NavigationSystem navigationSystem;  
    private readonly OperationLog operationLog;  
    private readonly IShip ship;  
    private readonly PortDispatcher portDispatcher;  
     
    public PortDispatcherTests()  
    {  
    engine = new Engine(200.0); *// Power > 100 to avoid Faulty state* cargoSystem = new CargoSystem(1000.0); *// Max capacity of 1000 tons* navigationSystem = new NavigationSystem();  
    operationLog = new OperationLog();  
    ship = new Ship("SHIP001", engine, cargoSystem, navigationSystem, operationLog);  
    portDispatcher = new PortDispatcher();  
    }  
     
    *// Original test: RequestReport\_ReturnsShipReport* [Fact]  
    public void RequestReport\_ReturnsShipReport()  
    {  
    *// Arrange* ship.Start(); *// Move to Waiting* ship.LoadCargo(500.0); *// Set cargo weight* ship.NavigateTo("Port B"); *// Set destination  
     
    // Act* var report = portDispatcher.RequestReport(ship);  
     
    *// Assert* Assert.Equal("Ship SHIP001: State=Moving, Cargo=500 tons, Destination=Port B", report);  
    }  
     
    *// Original test: SendCommand\_Navigate\_ChangesShipState* [Fact]  
    public void SendCommand\_Navigate\_ChangesShipState()  
    {  
    *// Arrange* ship.Start(); *// Переходим в Waiting  
     
    // Act* portDispatcher.SendCommand(ship, "navigate");  
     
    *// Assert* Assert.Equal(ShipState.**Moving**, ship.State);  
    Assert.Equal("Port B", navigationSystem.CurrentDestination);  
    Assert.Contains(operationLog.Logs, log => log.Contains("Navigating to Port B."));  
    }  
     
    *// Test for Sequence Diagram: Start Ship (Fig. 6.3)* [Fact]  
    public void StartShip\_ShouldTransitionToWaiting()  
    {  
    *// Arrange  
    // Ship is initially in Stopped state  
     
    // Act* portDispatcher.SendCommand(ship, "start");  
     
    *// Assert* Assert.Equal(ShipState.**Waiting**, ship.State);  
    Assert.Equal(EngineState.**On**, engine.State);  
    Assert.Contains(operationLog.Logs, log => log.Contains("Ship started and moved to Waiting state."));  
    }  
     
    *// Test for Sequence Diagram: Stop Ship (Fig. 6.4)* [Fact]  
    public void StopShip\_ShouldTransitionToStopped()  
    {  
    *// Arrange* portDispatcher.SendCommand(ship, "start"); *// Move to Waiting  
     
    // Act* portDispatcher.SendCommand(ship, "stop");  
     
    *// Assert* Assert.Equal(ShipState.**Stopped**, ship.State);  
    Assert.Equal(EngineState.**Off**, engine.State);  
    Assert.Contains(operationLog.Logs, log => log.Contains("Ship stopped."));  
    }  
     
    *// Test for Sequence Diagram: Perform Loading (Fig. 6.5)* [Fact]  
    public void PerformLoading\_ShouldLoadCargoAndReturnToWaiting()  
    {  
    *// Arrange* portDispatcher.SendCommand(ship, "start"); *// Move to Waiting  
     
    // Act* portDispatcher.SendCommand(ship, "load"); *// Loads 100 tons (per PortDispatcher)  
     
    // Assert* Assert.Equal(ShipState.**Waiting**, ship.State);  
    Assert.Equal(100.0, cargoSystem.CurrentWeight);  
    Assert.Equal(CargoState.**Empty**, cargoSystem.State); *// 100 tons < maxCapacity, so state remains Empty* Assert.Contains(operationLog.Logs, log => log.Contains("Loaded 100 tons of cargo."));  
    }  
     
    *// Test for Sequence Diagram: Perform Unloading (Fig. 6.6)* [Fact]  
    public void PerformUnloading\_ShouldUnloadCargoAndReturnToWaiting()  
    {  
    *// Arrange* portDispatcher.SendCommand(ship, "start"); *// Move to Waiting* portDispatcher.SendCommand(ship, "load"); *// Load 100 tons  
     
    // Act* portDispatcher.SendCommand(ship, "unload");  
     
    *// Assert* Assert.Equal(ShipState.**Waiting**, ship.State);  
    Assert.Equal(0.0, cargoSystem.CurrentWeight);  
    Assert.Equal(CargoState.**Empty**, cargoSystem.State);  
    Assert.Contains(operationLog.Logs, log => log.Contains("Cargo unloaded."));  
    }  
     
    *// Test for Sequence Diagram: Perform Navigation (Fig. 6.8)* [Fact]  
    public void PerformNavigation\_ShouldTransitionToMoving()  
    {  
    *// Arrange* portDispatcher.SendCommand(ship, "start"); *// Move to Waiting  
     
    // Act* portDispatcher.SendCommand(ship, "navigate");  
     
    *// Assert* Assert.Equal(ShipState.**Moving**, ship.State);  
    Assert.Equal("Port B", navigationSystem.CurrentDestination);  
    Assert.True(navigationSystem.IsActive);  
    Assert.Contains(operationLog.Logs, log => log.Contains("Navigating to Port B."));  
    }  
     
    *// Test for Sequence Diagram: Complete Navigation (Fig. 6.2)* [Fact]  
    public void CompleteNavigation\_ShouldReturnToWaiting()  
    {  
    *// Arrange* portDispatcher.SendCommand(ship, "start"); *// Move to Waiting* portDispatcher.SendCommand(ship, "navigate"); *// Move to Moving  
     
    // Act* ship.Wait(); *// Assuming Wait() is used to complete navigation, as no other method transitions Moving to Waiting  
     
    // Assert* Assert.Equal(ShipState.**Waiting**, ship.State);  
    Assert.Equal(EngineState.**On**, engine.State);  
    Assert.Contains(operationLog.Logs, log => log.Contains("Ship Waiting."));  
    }  
     
    *// Test for Sequence Diagram: Monitor State (Fig. 6.7)* [Fact]  
    public void MonitorState\_ShouldReturnCorrectReport()  
    {  
    *// Arrange* portDispatcher.SendCommand(ship, "start"); *// Move to Waiting* portDispatcher.SendCommand(ship, "load"); *// Load 100 tons* navigationSystem.SetDestination("Port C"); *// Manually set destination for test  
     
    // Act* var report = portDispatcher.RequestReport(ship);  
     
    *// Assert* Assert.Equal("Ship SHIP001: State=Waiting, Cargo=100 tons, Destination=Port C", report);  
    }  
     
    *// 1. Запуск судна и остановка (старт и стоп)* [Fact]  
    public void Scenario\_StartAndStopShip()  
    {  
    portDispatcher.SendCommand(ship, "start");  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    Assert.Equal(EngineState.**On**, engine.State);  
     
    portDispatcher.SendCommand(ship, "stop");  
    Assert.Equal(ShipState.**Stopped**, ship.State);  
    Assert.Equal(EngineState.**Off**, engine.State);  
    }  
     
    *// 2. Погрузка и разгрузка (используются: CargoSystem + Ship)* [Fact]  
    public void Scenario\_LoadAndUnloadCargo()  
    {  
    portDispatcher.SendCommand(ship, "start");  
    portDispatcher.SendCommand(ship, "load");  
    Assert.Equal(100.0, cargoSystem.CurrentWeight);  
    Assert.Contains(operationLog.Logs, l => l.Contains("Loaded 100 tons of cargo."));  
     
    portDispatcher.SendCommand(ship, "unload");  
    Assert.Equal(0.0, cargoSystem.CurrentWeight);  
    Assert.Contains(operationLog.Logs, l => l.Contains("Cargo unloaded."));  
    }  
     
    *// 3. Погрузка → Навигация (CargoSystem + NavigationSystem + Ship)* [Fact]  
    public void Scenario\_LoadCargoAndNavigate()  
    {  
    portDispatcher.SendCommand(ship, "start");  
    portDispatcher.SendCommand(ship, "load");  
    Assert.Equal(100.0, cargoSystem.CurrentWeight);  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
     
    portDispatcher.SendCommand(ship, "navigate");  
    Assert.Equal(ShipState.**Moving**, ship.State);  
    Assert.Equal("Port B", navigationSystem.CurrentDestination);  
    }  
     
    *// 4. Запуск → Навигация → Ожидание (Engine + NavigationSystem + Ship)* [Fact]  
    public void Scenario\_NavigateAndWait()  
    {  
    portDispatcher.SendCommand(ship, "start");  
    portDispatcher.SendCommand(ship, "navigate");  
    Assert.Equal(ShipState.**Moving**, ship.State);  
    Assert.Equal(EngineState.**On**, engine.State);  
     
    *// Возвращаемся в ожидание* ship.Wait();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    Assert.Equal(EngineState.**On**, engine.State);  
    }  
     
    *// 5. Интеграционный сценарий: затрагивает все объекты* [Fact]  
    public void Scenario\_FullOperation\_AllSubsystems()  
    {  
    *// 1. Запуск* portDispatcher.SendCommand(ship, "start");  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
     
    *// 2. Погрузка* portDispatcher.SendCommand(ship, "load");  
    Assert.Equal(100.0, cargoSystem.CurrentWeight);  
     
    *// 3. Навигация* portDispatcher.SendCommand(ship, "navigate");  
    Assert.Equal(ShipState.**Moving**, ship.State);  
     
    *// 4. Ожидание (завершить навигацию)* ship.Wait();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
     
    *// 5. Разгрузка* portDispatcher.SendCommand(ship, "unload");  
    Assert.Equal(0.0, cargoSystem.CurrentWeight);  
     
    *// 6. Останов* portDispatcher.SendCommand(ship, "stop");  
    Assert.Equal(ShipState.**Stopped**, ship.State);  
     
    *// Проверка логов и состояния всех объектов* Assert.Equal(EngineState.**Off**, engine.State);  
    Assert.Equal(CargoState.**Empty**, cargoSystem.State);  
    navigationSystem.StopNavigation();  
    Assert.Equal("None", navigationSystem.CurrentDestination);  
    Assert.Contains(operationLog.Logs, l => l.Contains("Ship started and moved to Waiting state."));  
    Assert.Contains(operationLog.Logs, l => l.Contains("Loaded 100 tons of cargo."));  
    Assert.Contains(operationLog.Logs, l => l.Contains("Navigating to Port B."));  
    Assert.Contains(operationLog.Logs, l => l.Contains("Cargo unloaded."));  
    Assert.Contains(operationLog.Logs, l => l.Contains("Ship stopped."));  
    }  
   }
   1. Тест «ShipTests.cs»
4. using ClassLibrary1.Interfaces;  
   using ClassLibrary1.Models;  
   using Moq;  
   using Xunit;  
     
   namespace TestProject2.UnitTests;  
     
   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);  
    }  
     
    *// --- PROPERTY TESTS ---  
     
    // Проверка корректного возврата идентификатора судна* [Fact]  
    public void Identifier\_ReturnsCorrectValue()  
    {  
    Assert.Equal("SHIP001", ship.Identifier);  
    }  
     
    *// Проверка, что судно изначально находится в состоянии Останов (Stopped)* [Fact]  
    public void State\_InitiallyStopped()  
    {  
    Assert.Equal(ShipState.**Stopped**, ship.State);  
    }  
     
    *// --- ОСНОВНЫЕ ПЕРЕХОДЫ СОСТОЯНИЙ ---  
     
    // 8. Останов → Ожидание  
    // Проверяет переход из состояния Останов в Ожидание через Start().* [Fact]  
    public void Start\_FromStopped\_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());  
    }  
     
    *// 9. Ожидание → Останов  
    // Проверяет переход из состояния Ожидание в Останов через Stop().* [Fact]  
    public void Stop\_FromWaiting\_ChangesStateToStopped()  
    {  
    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());  
    }  
     
    *// 14. Ожидание → Движение → Останов  
    // Проверяет переход из Ожидания в Движение через NavigateTo и далее в Останов через Stop.* [Fact]  
    public void Stop\_FromMoving\_ChangesStateToStopped()  
    {  
    ship.Start();  
    ship.NavigateTo("Port B");  
    ship.Stop();  
    Assert.Equal(ShipState.**Stopped**, ship.State);  
    engineMock.Verify(e => e.TurnOff(), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship stopped."), Times.Once());  
    }  
     
    *// 10. Ожидание → Погрузка → Ожидание  
    // Проверяет процесс погрузки грузов: из Ожидания вызывается LoadCargo, возвращается в Ожидание.* [Fact]  
    public void LoadCargo\_FromWaiting\_ChangesStateToLoadingThenWaiting()  
    {  
    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());  
    }  
     
    *// 3. Ожидание → Разгрузка → Ожидание  
    // Проверяет процесс разгрузки: из Ожидания вызывается UnloadCargo, возвращается в Ожидание.* [Fact]  
    public void UnloadCargo\_FromWaiting\_ChangesStateToUnloadingThenWaiting()  
    {  
    ship.Start();  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.UnloadCargo();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    cargoSystemMock.Verify(c => c.Unload(), Times.Once());  
    operationLogMock.Verify(l => l.Log("Cargo unloaded."), Times.Once());  
    }  
     
    *// 11. Ожидание → Движение  
    // Проверяет переход из Ожидания в Движение через NavigateTo.* [Fact]  
    public void NavigateTo\_FromWaiting\_ChangesStateToMoving()  
    {  
    ship.Start();  
    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());  
    }  
     
    *// 12. Движение → Ожидание  
    // Проверяет возвращение из Движения в Ожидание через Wait.* [Fact]  
    public void Wait\_FromMoving\_ChangesStateToWaiting()  
    {  
    ship.Start();  
    ship.NavigateTo("Port B");  
    ship.Wait();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    engineMock.Verify(e => e.TurnOn(), Times.Exactly(2)); *// Start and Wait* operationLogMock.Verify(l => l.Log("Ship Waiting."), Times.Once());  
    }  
     
    *// 13. Останов → Ожидание (через Wait)  
    // Проверяет переход в Ожидание из Останов через Wait.* [Fact]  
    public void Wait\_FromStopped\_ChangesStateToWaiting()  
    {  
    ship.Wait();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    engineMock.Verify(e => e.TurnOn(), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship Waiting."), Times.Once());  
    }  
     
    *// --- СЛОЖНЫЕ СЦЕНАРИИ (СЕКВЕНЦИИ) ---  
     
    // 1. Останов → Ожидание → Останов  
    // Проверяет базовый цикл запуска и остановки.* [Fact]  
    public void Scenario\_StoppedToWaitingToStopped()  
    {  
    ship.Start();  
    ship.Stop();  
    Assert.Equal(ShipState.**Stopped**, ship.State);  
    engineMock.Verify(e => e.TurnOn(), Times.Once());  
    engineMock.Verify(e => e.TurnOff(), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship started and moved to Waiting state."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship stopped."), Times.Once());  
    }  
     
    *// 2. Ожидание → Погрузка → Ожидание  
    // Проверяет процесс погрузки грузов (секвенция).* [Fact]  
    public void Scenario\_WaitingToLoadingToWaiting()  
    {  
    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());  
    }  
     
    *// 3. Ожидание → Разгрузка → Ожидание  
    // Проверяет процесс разгрузки грузов (секвенция).* [Fact]  
    public void Scenario\_WaitingToUnloadingToWaiting()  
    {  
    ship.Start();  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.UnloadCargo();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    cargoSystemMock.Verify(c => c.Unload(), Times.Once());  
    operationLogMock.Verify(l => l.Log("Cargo unloaded."), Times.Once());  
    }  
     
    *// 4. Ожидание → Движение → Ожидание  
    // Проверяет процесс навигации и возврата в Ожидание.* [Fact]  
    public void Scenario\_WaitingToMovingToWaiting()  
    {  
    ship.Start();  
    ship.NavigateTo("Port B");  
    ship.Wait();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    navigationSystemMock.Verify(n => n.SetDestination("Port B"), Times.Once());  
    engineMock.Verify(e => e.TurnOn(), Times.Exactly(2)); *// Start and Wait* operationLogMock.Verify(l => l.Log("Navigating to Port B."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship Waiting."), Times.Once());  
    }  
     
    *// 5. Ожидание → Погрузка → Ожидание → Движение → Ожидание  
    // Проверяет комбинированный сценарий погрузки и последующей навигации.* [Fact]  
    public void Scenario\_WaitingToLoadingToWaitingToMovingToWaiting()  
    {  
    ship.Start();  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.LoadCargo(100.0);  
    ship.NavigateTo("Port C");  
    ship.Wait();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    cargoSystemMock.Verify(c => c.Load(100.0), Times.Once());  
    navigationSystemMock.Verify(n => n.SetDestination("Port C"), Times.Once());  
    engineMock.Verify(e => e.TurnOn(), Times.Exactly(2)); *// Start and Wait* operationLogMock.Verify(l => l.Log("Loaded 100 tons of cargo."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Navigating to Port C."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship Waiting."), Times.Once());  
    }  
     
    *// 6. Ожидание → Разгрузка → Ожидание → Движение → Ожидание  
    // Проверяет комбинированный сценарий разгрузки и последующей навигации.* [Fact]  
    public void Scenario\_WaitingToUnloadingToWaitingToMovingToWaiting()  
    {  
    ship.Start();  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.UnloadCargo();  
    ship.NavigateTo("Port C");  
    ship.Wait();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    cargoSystemMock.Verify(c => c.Unload(), Times.Once());  
    navigationSystemMock.Verify(n => n.SetDestination("Port C"), Times.Once());  
    engineMock.Verify(e => e.TurnOn(), Times.Exactly(2)); *// Start and Wait* operationLogMock.Verify(l => l.Log("Cargo unloaded."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Navigating to Port C."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship Waiting."), Times.Once());  
    }  
     
    *// 7. Ожидание → Погрузка → Ожидание → Разгрузка → Ожидание → Движение → Ожидание  
    // Проверяет полный цикл операций: погрузка, разгрузка, навигация.* [Fact]  
    public void Scenario\_WaitingToLoadingToWaitingToUnloadingToWaitingToMovingToWaiting()  
    {  
    ship.Start();  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.LoadCargo(100.0);  
    ship.UnloadCargo();  
    ship.NavigateTo("Port C");  
    ship.Wait();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    cargoSystemMock.Verify(c => c.Load(100.0), Times.Once());  
    cargoSystemMock.Verify(c => c.Unload(), Times.Once());  
    navigationSystemMock.Verify(n => n.SetDestination("Port C"), Times.Once());  
    engineMock.Verify(e => e.TurnOn(), Times.Exactly(2)); *// Start and Wait* operationLogMock.Verify(l => l.Log("Loaded 100 tons of cargo."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Cargo unloaded."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Navigating to Port C."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship Waiting."), Times.Once());  
    }  
     
    *// --- ДОПОЛНИТЕЛЬНЫЕ/EDGE CASE СЕКВЕНЦИИ ---  
     
    // Повторные загрузки: Ожидание → Погрузка → Ожидание → Погрузка → Ожидание* [Fact]  
    public void Scenario\_WaitingToLoadingToWaitingToLoadingToWaiting()  
    {  
    ship.Start();  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.LoadCargo(100.0);  
    ship.LoadCargo(100.0);  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    cargoSystemMock.Verify(c => c.Load(100.0), Times.Exactly(2));  
    operationLogMock.Verify(l => l.Log("Loaded 100 tons of cargo."), Times.Exactly(2));  
    }  
     
    *// Повторные разгрузки: Ожидание → Разгрузка → Ожидание → Разгрузка → Ожидание* [Fact]  
    public void Scenario\_WaitingToUnloadingToWaitingToUnloadingToWaiting()  
    {  
    ship.Start();  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.UnloadCargo();  
    ship.UnloadCargo();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    cargoSystemMock.Verify(c => c.Unload(), Times.Exactly(2));  
    operationLogMock.Verify(l => l.Log("Cargo unloaded."), Times.Exactly(2));  
    }  
     
    *// Повторная навигация: Ожидание → Движение → Ожидание → Движение → Ожидание* [Fact]  
    public void Scenario\_WaitingToMovingToWaitingToMovingToWaiting()  
    {  
    ship.Start();  
    ship.NavigateTo("Port B");  
    ship.Wait();  
    ship.NavigateTo("Port C");  
    ship.Wait();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    navigationSystemMock.Verify(n => n.SetDestination("Port B"), Times.Once());  
    navigationSystemMock.Verify(n => n.SetDestination("Port C"), Times.Once());  
    engineMock.Verify(e => e.TurnOn(), Times.Exactly(3)); *// Start and two Wait calls* operationLogMock.Verify(l => l.Log("Navigating to Port B."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Navigating to Port C."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship Waiting."), Times.Exactly(2));  
    }  
     
    *// Запуск, остановка, повторный запуск, движение, ожидание* [Fact]  
    public void Scenario\_StoppedToWaitingToStoppedToWaitingToMovingToWaiting()  
    {  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.Start();  
    ship.Stop();  
    ship.Start();  
    ship.NavigateTo("Port B");  
    ship.Wait();  
    Assert.Equal(ShipState.**Waiting**, ship.State);  
    engineMock.Verify(e => e.TurnOn(), Times.Exactly(3)); *// Two Start and one Wait* engineMock.Verify(e => e.TurnOff(), Times.Once());  
    navigationSystemMock.Verify(n => n.SetDestination("Port B"), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship started and moved to Waiting state."), Times.Exactly(2));  
    operationLogMock.Verify(l => l.Log("Ship stopped."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Navigating to Port B."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship Waiting."), Times.Once());  
    }  
     
    *// --- НЕВАЛИДНЫЕ ПЕРЕХОДЫ ---  
     
    // Проверяет, что нельзя остановить судно в состоянии Останов* [Fact]  
    public void Stop\_FromStopped\_DoesNotChangeState()  
    {  
    var initialState = ship.State;  
    ship.Stop();  
    Assert.Equal(initialState, ship.State);  
    engineMock.Verify(e => e.TurnOff(), Times.Never());  
    operationLogMock.Verify(l => l.Log(It.IsAny<string>()), Times.Never());  
    }  
     
    *// Проверяет, что нельзя загрузить в состоянии Останов* [Fact]  
    public void LoadCargo\_FromStopped\_DoesNotChangeState()  
    {  
    var initialState = ship.State;  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.LoadCargo(100.0);  
    Assert.Equal(initialState, ship.State);  
    cargoSystemMock.Verify(c => c.Load(It.IsAny<double>()), Times.Never());  
    operationLogMock.Verify(l => l.Log(It.IsAny<string>()), Times.Never());  
    }  
     
    *// Проверяет, что нельзя разгрузить в состоянии Останов* [Fact]  
    public void UnloadCargo\_FromStopped\_DoesNotChangeState()  
    {  
    var initialState = ship.State;  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.UnloadCargo();  
    Assert.Equal(initialState, ship.State);  
    cargoSystemMock.Verify(c => c.Unload(), Times.Never());  
    operationLogMock.Verify(l => l.Log(It.IsAny<string>()), Times.Never());  
    }  
     
    *// Проверяет, что нельзя начать навигацию в состоянии Останов* [Fact]  
    public void NavigateTo\_FromStopped\_DoesNotChangeState()  
    {  
    var initialState = ship.State;  
    ship.NavigateTo("Port B");  
    Assert.Equal(initialState, ship.State);  
    navigationSystemMock.Verify(n => n.SetDestination(It.IsAny<string>()), Times.Never());  
    operationLogMock.Verify(l => l.Log(It.IsAny<string>()), Times.Never());  
    }  
     
    *// Проверяет, что нельзя повторно запустить судно в Ожидании* [Fact]  
    public void Start\_FromWaiting\_DoesNotChangeState()  
    {  
    ship.Start();  
    var initialState = ship.State;  
    ship.Start();  
    Assert.Equal(initialState, ship.State);  
    engineMock.Verify(e => e.TurnOn(), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship started and moved to Waiting state."), Times.Once());  
    }  
     
    *// Проверяет, что повторный вызов Wait в Ожидании не влияет на состояние* [Fact]  
    public void Wait\_FromWaiting\_DoesNotInvokeEngineOrLog()  
    {  
    ship.Start();  
    var initialState = ship.State;  
    ship.Wait();  
    Assert.Equal(initialState, ship.State);  
    engineMock.Verify(e => e.TurnOn(), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship Waiting."), Times.Never());  
    }  
     
    *// Проверяет, что нельзя запустить судно в Движении* [Fact]  
    public void Start\_FromMoving\_DoesNotChangeState()  
    {  
    ship.Start();  
    ship.NavigateTo("Port B");  
    var initialState = ship.State;  
    ship.Start();  
    Assert.Equal(initialState, ship.State);  
    engineMock.Verify(e => e.TurnOn(), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship started and moved to Waiting state."), Times.Once());  
    }  
     
    *// Проверяет, что нельзя загрузить груз в Движении* [Fact]  
    public void LoadCargo\_FromMoving\_DoesNotChangeState()  
    {  
    ship.Start();  
    ship.NavigateTo("Port B");  
    var initialState = ship.State;  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.LoadCargo(100.0);  
    Assert.Equal(initialState, ship.State);  
    cargoSystemMock.Verify(c => c.Load(It.IsAny<double>()), Times.Never());  
    operationLogMock.Verify(l => l.Log("Loaded 100 tons of cargo."), Times.Never());  
    }  
     
    *// Проверяет, что нельзя разгрузить груз в Движении* [Fact]  
    public void UnloadCargo\_FromMoving\_DoesNotChangeState()  
    {  
    ship.Start();  
    ship.NavigateTo("Port B");  
    var initialState = ship.State;  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.UnloadCargo();  
    Assert.Equal(initialState, ship.State);  
    cargoSystemMock.Verify(c => c.Unload(), Times.Never());  
    operationLogMock.Verify(l => l.Log("Cargo unloaded."), Times.Never());  
    }  
     
    *// Проверяет, что нельзя начать другую навигацию в Движении* [Fact]  
    public void NavigateTo\_FromMoving\_DoesNotChangeState()  
    {  
    ship.Start();  
    ship.NavigateTo("Port B");  
    var initialState = ship.State;  
    ship.NavigateTo("Port C");  
    Assert.Equal(initialState, ship.State);  
    navigationSystemMock.Verify(n => n.SetDestination("Port B"), Times.Once());  
    navigationSystemMock.Verify(n => n.SetDestination("Port C"), Times.Never());  
    operationLogMock.Verify(l => l.Log("Navigating to Port C."), Times.Never());  
    }  
     
    *// --- ПРОЧИЕ ТЕСТЫ ---  
     
    // Проверка, что генерация отчёта не меняет состояние* [Fact]  
    public void GenerateReport\_DoesNotChangeState()  
    {  
    ship.Start();  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(500.0);  
    navigationSystemMock.Setup(n => n.CurrentDestination).Returns("Port B");  
    var initialState = ship.State;  
    var report = ship.GenerateReport();  
    Assert.Equal(initialState, ship.State);  
    Assert.Equal("Ship SHIP001: State=Waiting, Cargo=500 tons, Destination=Port B", report);  
    operationLogMock.Verify(l => l.Log(It.IsAny<string>()), Times.Once()); *// Only from Start* }  
      
    *// 14. Ожидание → Движение → Останов (ещё раз, как сценарий)* [Fact]  
    public void Scenario\_WaitingToMovingToStopped()  
    {  
    ship.Start();  
    cargoSystemMock.Setup(c => c.CurrentWeight).Returns(100.0);  
    ship.NavigateTo("Port B");  
    ship.Stop();  
    Assert.Equal(ShipState.**Stopped**, ship.State);  
    navigationSystemMock.Verify(n => n.SetDestination("Port B"), Times.Once());  
    engineMock.Verify(e => e.TurnOn(), Times.Once());  
    engineMock.Verify(e => e.TurnOff(), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship started and moved to Waiting state."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Navigating to Port B."), Times.Once());  
    operationLogMock.Verify(l => l.Log("Ship stopped."), Times.Once());  
    }  
   }