МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по лабораторной работе №6-7
“ Шаблоны проектирования и модульное тестирование в Python.”

Выполнил:
Студент группы ИУ5Ц-41Б
Цурин А.П,
Преподаватель:
Гапанюк Ю.Е.

Москва 2025

1. Задания для выполнения

Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог. Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.

В модульных тестах необходимо применить следующие технологии:

TDD - фреймворк.

BDD - фреймворк.

Создание Mock-объектов.

2. Листинг программы

Features/steps/library_steps.py

```
"""
BDD шаги для библиотечной системы
"""
from behave import given, when, then
from library_system import *


@given('библиотечная система')
def step_create_library(context):
    context.library = Library()


@given('фабрика документов')
def step_create_factory(context):
    context.factory = DocumentFactory()


@given('адаптер для старой системы')
def step_create_adapter(context):
    context.old_system = OldLibrarySystem()
    context.adapter = LibraryAdapter(context.old_system)


@given('email уведомления для "{email}"')
def step_create_email_notifier(context, email):
    context.email_notifier = EmailNotifier(email)
    context.library.add_observer(context.email_notifier)


@given('документ "{doc_type}" с названием "{title}" автора "{author}" {year:d} года')
def step_create_document(context, doc_type, title, author, year):
    context.document = context.factory.create_document(doc_type, title, author, year)


@given('добавлен документ "{doc_type}" с названием "{title}" автора "{author}" {year:d}
года')
def step_add_document_to_library(context, doc_type, title, author, year):
    context.library.add_document(doc_type, title, author, year)
```

```python
@when('я создаю документ "{doc_type}" с названием "{title}" автора "{author}" {year:d} года')
def step_create_document_action(context, doc_type, title, author, year):
    context.document = context.factory.create_document(doc_type, title, author, year)


@when('я добавляю документ в библиотеку')
def step_add_document_to_library_action(context):
    context.library.documents.append(context.document)


@when('пользователь "{user}" берет документ "{title}"')
def step_borrow_document(context, user, title):
    context.borrow_result = context.library.borrow_document(title, user)


@when('пользователь "{user}" возвращает документ "{title}"')
def step_return_document(context, user, title):
    context.return_result = context.library.return_document(title, user)


@when('я добавляю документ через адаптер')
def step_add_document_via_adapter(context):
    context.doc_id = context.adapter.add_document(context.document)


@then('документ должен быть типа "{doc_type}"')
def step_check_document_type(context, doc_type):
    assert context.document.get_type() == doc_type


@then('период выдачи должен быть {days:d} дней')
def step_check_loan_period(context, days):
    assert context.document.get_loan_period() == days


@then('документ должен быть доступен')
def step_check_document_available(context):
    assert context.document.is_available()


@then('документ должен быть недоступен')
def step_check_document_unavailable(context):
    assert not context.document.is_available()


@then('операция выдачи должна быть успешной')
def step_check_borrow_success(context):
    assert context.borrow_result is True


@then('операция возврата должна быть успешной')
def step_check_return_success(context):
```

```
    assert context.return_result is True


@then('я должен получить ID документа')
def step_check_document_id(context):
    assert hasattr(context, 'doc_id')
    assert isinstance(context.doc_id, int)


@then('уведомление должно быть отправлено')
def step_check_notification_sent(context):
    assert len(context.email_notifier.notifications) > 0
```

features/library.feature

```
# language: ru
Функционал: Библиотечная система с шаблонами проектирования


  Контекст:
    Как пользователь библиотеки
    Я хочу использовать библиотечную систему
    Чтобы управлять документами и получать уведомления


  Сценарий: Создание документа через фабрику
    Дано фабрика документов
    Когда я создаю документ "book" с названием "Python Programming" автора "John Doe" 2023 года
    Тогда документ должен быть типа "Book"
    И период выдачи должен быть 30 дней


  Сценарий: Выдача и возврат документа
    Дано библиотечная система
    И добавлен документ "book" с названием "Test Book" автора "Test Author" 2023 года
    Когда пользователь "user123" берет документ "Test Book"
    Тогда операция выдачи должна быть успешной
    И документ должен быть недоступен
    И когда пользователь "user123" возвращает документ "Test Book"
    Тогда операция возврата должна быть успешной
    И документ должен быть доступен


  Сценарий: Работа адаптера
    Дано адаптер для старой системы
    И документ "book" с названием "Adapter Test" автора "Test Author" 2023 года
```

Когда я добавляю документ через адаптер

Тогда я должен получить ID документа

Сценарий: Система уведомлений

Дано библиотечная система

И email уведомления для "user@example.com"

Когда я добавляю документ "book" с названием "Notification Test" автора "Author" 2023 года

Тогда уведомление должно быть отправлено

Demo.py

```
"""
Демонстрация работы библиотечной системы
"""
from library_system import *


def main():
    print(" ДЕМОНСТРАЦИЯ БИБЛИОТЕЧНОЙ СИСТЕМЫ")
    print("=" * 50)

    # Создание библиотеки
    library = Library()

    # Добавление наблюдателей
    email_notifier = EmailNotifier("admin@library.ru")
    sms_notifier = SMSNotifier("+7-999-123-45-67")
    logger = Logger()

    library.add_observer(email_notifier)
    library.add_observer(sms_notifier)
    library.add_observer(logger)

    print("\n1. СОЗДАНИЕ ДОКУМЕНТОВ ЧЕРЕЗ ФАБРИКУ:")
    print("-" * 40)

    # Создание документов разных типов
    book = library.add_document("book", "Python Programming", "John Doe", 2023)
    magazine = library.add_document("magazine", "Science Today", "Editor", 2024)
    article = library.add_document("article", "AI Research", "Dr. Smith", 2023)

    print(f" Создана книга: {book.title}")
    print(f" Создан журнал: {magazine.title}")
    print(f" Создана статья: {article.title}")

    print("\n2. РАБОТА АДАПТЕРА:")
    print("-" * 40)

    # Демонстрация работы адаптера
    doc_id = library.adapter.add_document(book)
```

```
    doc_info = library.adapter.get_document(doc_id)

    print(f" ID документа в старой системе: {doc_id}")
    print(f" Информация: {doc_info['title']} ({doc_info['type']})")
    print(f" Доступен: {doc_info['available']}")

    print("\n3. ВЫДАЧА И ВОЗВРАТ ДОКУМЕНТОВ:")
    print("-" * 40)

    # Workflow выдачи и возврата
    print(" Выдача книги...")
    library.borrow_document("Python Programming", "user123")

    print(" Возврат книги...")
    library.return_document("Python Programming", "user123")

    print("\n4. СТАТИСТИКА:")
    print("-" * 40)

    print(f" Всего документов: {len(library.documents)}")
    print(f" Доступно: {len(library.get_available_documents())}")
    print(f" Email уведомлений: {len(email_notifier.notifications)}")
    print(f" SMS уведомлений: {len(sms_notifier.notifications)}")
    print(f" Логов: {len(logger.logs)}")

    print("\n" + "=" * 50)
    print(" Демонстрация завершена успешно!")


if __name__ == "__main__":
    main()
```

library_system.py

```
"""
Библиотечная система с шаблонами проектирования
"""
from abc import ABC, abstractmethod
from typing import List, Dict, Any
from datetime import datetime
import random


# ==================== ПОРОЖДАЮЩИЙ: Фабричный метод ====================

class Document(ABC):
    """Абстрактный класс документа"""

    def __init__(self, title: str, author: str, year: int):
        self.title = title
        self.author = author
        self.year = year
        self._is_borrowed = False
```

```python
    @abstractmethod
    def get_type(self) -> str:
        pass

    @abstractmethod
    def get_loan_period(self) -> int:
        pass

    def borrow(self) -> bool:
        if not self._is_borrowed:
            self._is_borrowed = True
            return True
        return False

    def return_doc(self) -> bool:
        if self._is_borrowed:
            self._is_borrowed = False
            return True
        return False

    def is_available(self) -> bool:
        return not self._is_borrowed

    def get_info(self) -> Dict[str, Any]:
        return {
            "type": self.get_type(),
            "title": self.title,
            "author": self.author,
            "year": self.year,
            "available": self.is_available(),
            "loan_period": self.get_loan_period()
        }


class Book(Document):
    """Книга"""

    def get_type(self) -> str:
        return "Book"

    def get_loan_period(self) -> int:
        return 30


class Magazine(Document):
    """Журнал"""

    def get_type(self) -> str:
        return "Magazine"

    def get_loan_period(self) -> int:
        return 14
```

```python
class Article(Document):
    """Статья"""

    def get_type(self) -> str:
        return "Article"

    def get_loan_period(self) -> int:
        return 7


class DocumentFactory:
    """Фабрика документов"""

    @staticmethod
    def create_document(doc_type: str, title: str, author: str, year: int) -> Document:
        doc_type = doc_type.lower()
        if doc_type == "book":
            return Book(title, author, year)
        elif doc_type == "magazine":
            return Magazine(title, author, year)
        elif doc_type == "article":
            return Article(title, author, year)
        else:
            raise ValueError(f"Unknown document type: {doc_type}")


# ==================== СТРУКТУРНЫЙ: Адаптер ====================

class OldLibrarySystem:
    """Старая библиотечная система (несовместимый интерфейс)"""

    def __init__(self):
        self._storage = {}

    def store_document(self, doc_id: int, doc_data: dict) -> None:
        """Сохраняет документ в старом формате"""
        self._storage[doc_id] = doc_data

    def retrieve_document(self, doc_id: int) -> dict:
        """Извлекает документ в старом формате"""
        return self._storage.get(doc_id, {})

    def check_status_old(self, doc_id: int) -> str:
        """Проверяет статус в старом формате"""
        doc = self._storage.get(doc_id)
        if not doc:
            return "not_exists"
        return "free" if doc.get("is_free", True) else "taken"


class LibraryAdapter:
    """Адаптер для совместимости со старой системой"""
```

```python
    def __init__(self, old_system: OldLibrarySystem):
        self.old_system = old_system
        self._next_id = 1

    def add_document(self, document: Document) -> int:
        """Добавляет документ через адаптер"""
        doc_id = self._next_id
        self._next_id += 1

        # Преобразование в старый формат
        old_format_data = {
            "name": document.title,
            "creator": document.author,
            "year_published": document.year,
            "doc_type": document.get_type(),
            "is_free": document.is_available()
        }

        self.old_system.store_document(doc_id, old_format_data)
        return doc_id

    def get_document(self, doc_id: int) -> Dict[str, Any]:
        """Получает документ в современном формате"""
        old_data = self.old_system.retrieve_document(doc_id)
        if not old_data:
            return {}

        # Преобразование в современный формат
        return {
            "id": doc_id,
            "title": old_data.get("name", ""),
            "author": old_data.get("creator", ""),
            "year": old_data.get("year_published", 0),
            "type": old_data.get("doc_type", "unknown"),
            "available": old_data.get("is_free", False)
        }

    def is_available(self, doc_id: int) -> bool:
        """Проверяет доступность через адаптер"""
        status = self.old_system.check_status_old(doc_id)
        return status == "free"


# ==================== ПОВЕДЕНЧЕСКИЙ: Наблюдатель ====================

class LibraryEvent:
    """Событие в библиотеке"""

    def __init__(self, event_type: str, document: Document, user: str = None):
        self.type = event_type
        self.document = document
        self.user = user
```

```
        self.timestamp = datetime.now()

    def __str__(self):
        user_info = f" by {self.user}" if self.user else ""
        return f"{self.timestamp}: {self.type} - '{self.document.title}'{user_info}"


class Observer(ABC):
    """Абстрактный наблюдатель"""

    @abstractmethod
    def update(self, event: LibraryEvent) -> None:
        pass


class Subject:
    """Субъект для наблюдения"""

    def __init__(self):
        self._observers: List[Observer] = []

    def attach(self, observer: Observer) -> None:
        self._observers.append(observer)

    def detach(self, observer: Observer) -> None:
        if observer in self._observers:
            self._observers.remove(observer)

    def notify(self, event: LibraryEvent) -> None:
        for observer in self._observers:
            observer.update(event)


class EmailNotifier(Observer):
    """Email уведомления"""

    def __init__(self, email: str):
        self.email = email
        self.notifications = []

    def update(self, event: LibraryEvent) -> None:
        message = f"Email to {self.email}: {event}"
        self.notifications.append(message)
        print(message)


class SMSNotifier(Observer):
    """SMS уведомления"""

    def __init__(self, phone: str):
        self.phone = phone
        self.notifications = []

    def update(self, event: LibraryEvent) -> None:
```

```
        message = f"SMS to {self.phone}: {event.type} - {event.document.title}"
        self.notifications.append(message)
        print(message)


class Logger(Observer):
    """Логгер событий"""

    def __init__(self):
        self.logs = []

    def update(self, event: LibraryEvent) -> None:
        log_entry = f"LOG: {event}"
        self.logs.append(log_entry)
        print(log_entry)


# ==================== ОСНОВНАЯ СИСТЕМА ====================

class Library:
    """Основная библиотечная система"""

    def __init__(self):
        self.documents: List[Document] = []
        self.factory = DocumentFactory()
        self.old_system = OldLibrarySystem()
        self.adapter = LibraryAdapter(self.old_system)
        self.notifier = Subject()

    def add_document(self, doc_type: str, title: str, author: str, year: int) ->
Document:
        """Добавляет документ в библиотеку"""
        document = self.factory.create_document(doc_type, title, author, year)
        self.documents.append(document)

        # Адаптация для старой системы
        self.adapter.add_document(document)

        # Уведомление наблюдателей
        event = LibraryEvent("DOCUMENT_ADDED", document)
        self.notifier.notify(event)

        return document

    def borrow_document(self, title: str, user: str) -> bool:
        """Выдает документ пользователю"""
        for doc in self.documents:
            if doc.title == title and doc.is_available():
                if doc.borrow():
                    event = LibraryEvent("DOCUMENT_BORROWED", doc, user)
                    self.notifier.notify(event)
                    return True
        return False
```

```
    def return_document(self, title: str, user: str) -> bool:
        """Возвращает документ в библиотеку"""
        for doc in self.documents:
            if doc.title == title and not doc.is_available():
                if doc.return_doc():
                    event = LibraryEvent("DOCUMENT_RETURNED", doc, user)
                    self.notifier.notify(event)
                    return True
        return False

    def get_available_documents(self) -> List[Document]:
        """Возвращает список доступных документов"""
        return [doc for doc in self.documents if doc.is_available()]

    def get_document_info(self, title: str) -> Dict[str, Any]:
        """Возвращает информацию о документе"""
        for doc in self.documents:
            if doc.title == title:
                return doc.get_info()
        return {}

    def add_observer(self, observer: Observer) -> None:
        """Добавляет наблюдателя"""
        self.notifier.attach(observer)
```

test_library.py

```
"""
Модульные тесты для библиотечной системы
"""
import unittest
from unittest.mock import Mock, MagicMock, patch
from library_system import *


class TestDocumentFactory(unittest.TestCase):
    """Тесты фабрики документов"""

    def test_create_book(self):
        book = DocumentFactory.create_document("book", "Test Book", "Author", 2023)
        self.assertIsInstance(book, Book)
        self.assertEqual(book.get_type(), "Book")
        self.assertEqual(book.get_loan_period(), 30)

    def test_create_magazine(self):
        magazine = DocumentFactory.create_document("magazine", "Test Magazine",
"Editor", 2023)
        self.assertIsInstance(magazine, Magazine)
        self.assertEqual(magazine.get_type(), "Magazine")
        self.assertEqual(magazine.get_loan_period(), 14)

    def test_create_article(self):
```

```
        article = DocumentFactory.create_document("article", "Test Article",
"Researcher", 2023)
        self.assertIsInstance(article, Article)
        self.assertEqual(article.get_type(), "Article")
        self.assertEqual(article.get_loan_period(), 7)

    def test_invalid_document_type(self):
        with self.assertRaises(ValueError):
            DocumentFactory.create_document("invalid", "Test", "Author", 2023)


class TestDocumentMethods(unittest.TestCase):
    """Тесты методов документов"""

    def setUp(self):
        self.book = Book("Python Programming", "John Doe", 2023)

    def test_borrow_and_return(self):
        self.assertTrue(self.book.is_available())

        self.assertTrue(self.book.borrow())
        self.assertFalse(self.book.is_available())

        self.assertTrue(self.book.return_doc())
        self.assertTrue(self.book.is_available())

    def test_double_borrow(self):
        self.assertTrue(self.book.borrow())
        self.assertFalse(self.book.borrow())  # Нельзя взять уже занятую книгу

    def test_get_info(self):
        info = self.book.get_info()
        self.assertEqual(info["title"], "Python Programming")
        self.assertEqual(info["author"], "John Doe")
        self.assertEqual(info["type"], "Book")
        self.assertTrue(info["available"])


class TestAdapterPattern(unittest.TestCase):
    """Тесты паттерна Адаптер"""

    def setUp(self):
        self.old_system = OldLibrarySystem()
        self.adapter = LibraryAdapter(self.old_system)
        self.book = Book("Adapter Test", "Test Author", 2023)

    def test_add_document_via_adapter(self):
        doc_id = self.adapter.add_document(self.book)

        self.assertIsInstance(doc_id, int)

        old_data = self.old_system.retrieve_document(doc_id)
        self.assertEqual(old_data["name"], "Adapter Test")
```

```
        self.assertEqual(old_data["creator"], "Test Author")

    def test_get_document_via_adapter(self):
        doc_id = self.adapter.add_document(self.book)
        modern_data = self.adapter.get_document(doc_id)

        self.assertEqual(modern_data["title"], "Adapter Test")
        self.assertEqual(modern_data["author"], "Test Author")
        self.assertEqual(modern_data["type"], "Book")

    def test_check_availability(self):
        doc_id = self.adapter.add_document(self.book)
        self.assertTrue(self.adapter.is_available(doc_id))

        self.book.borrow()
        doc_id2 = self.adapter.add_document(self.book)
        self.assertFalse(self.adapter.is_available(doc_id2))


class TestObserverPattern(unittest.TestCase):
    """Тесты паттерна Наблюдатель"""

    def setUp(self):
        self.subject = Subject()
        self.email_observer = EmailNotifier("test@example.com")
        self.sms_observer = SMSNotifier("+1234567890")
        self.book = Book("Observer Test", "Author", 2023)

    def test_attach_and_notify(self):
        self.subject.attach(self.email_observer)
        self.subject.attach(self.sms_observer)

        event = LibraryEvent("TEST_EVENT", self.book, "user123")
        self.subject.notify(event)

        self.assertEqual(len(self.email_observer.notifications), 1)
        self.assertEqual(len(self.sms_observer.notifications), 1)

    def test_detach_observer(self):
        self.subject.attach(self.email_observer)
        self.subject.attach(self.sms_observer)

        self.subject.detach(self.email_observer)

        event = LibraryEvent("TEST_EVENT", self.book)
        self.subject.notify(event)

        self.assertEqual(len(self.email_observer.notifications), 0)
        self.assertEqual(len(self.sms_observer.notifications), 1)


class TestLibraryIntegration(unittest.TestCase):
    """Интеграционные тесты библиотеки"""
```

```
    def setUp(self):
        self.library = Library()
        self.email_notifier = EmailNotifier("user@example.com")
        self.library.add_observer(self.email_notifier)

    def test_add_document(self):
        document = self.library.add_document("book", "Integration Test", "Author",
2023)

        self.assertIsInstance(document, Book)
        self.assertEqual(len(self.library.documents), 1)
        self.assertEqual(len(self.email_notifier.notifications), 1)

    def test_borrow_return_workflow(self):
        document = self.library.add_document("book", "Workflow Test", "Author", 2023)

        # Выдача
        borrow_result = self.library.borrow_document("Workflow Test", "user123")
        self.assertTrue(borrow_result)
        self.assertFalse(document.is_available())

        # Возврат
        return_result = self.library.return_document("Workflow Test", "user123")
        self.assertTrue(return_result)
        self.assertTrue(document.is_available())

    def test_get_available_documents(self):
        self.library.add_document("book", "Available Book", "Author", 2023)
        self.library.add_document("magazine", "Borrowed Magazine", "Editor", 2023)

        # Занимаем один документ
        borrowed_doc = self.library.documents[1]
        borrowed_doc.borrow()

        available = self.library.get_available_documents()
        self.assertEqual(len(available), 1)
        self.assertEqual(available[0].title, "Available Book")

    def test_get_document_info(self):
        self.library.add_document("book", "Info Test", "Author", 2023)

        info = self.library.get_document_info("Info Test")
        self.assertEqual(info["title"], "Info Test")
        self.assertEqual(info["author"], "Author")
        self.assertEqual(info["type"], "Book")
        self.assertTrue(info["available"])


if __name__ == '__main__':
    unittest.main()
```

test_mocks.py

```python
"""
Тесты с использованием Mock объектов
"""
import unittest
from unittest.mock import Mock, MagicMock, patch
from library_system import *


class TestMockObjects(unittest.TestCase):
    """Тесты с mock объектами"""

    def test_mock_document(self):
        """Тест с mock документом"""
        mock_doc = MagicMock()
        mock_doc.title = "Mock Book"
        mock_doc.is_available.return_value = True
        mock_doc.borrow.return_value = True

        library = Library()
        library.documents = [mock_doc]

        result = library.borrow_document("Mock Book", "user123")
        self.assertTrue(result)
        mock_doc.borrow.assert_called_once()

    def test_mock_observer(self):
        """Тест с mock наблюдателем"""
        mock_observer = Mock(spec=Observer)
        subject = Subject()
        subject.attach(mock_observer)

        book = Book("Test Book", "Author", 2023)
        event = LibraryEvent("TEST_EVENT", book)

        subject.notify(event)
        mock_observer.update.assert_called_once_with(event)

    @patch('library_system.DocumentFactory.create_document')
    def test_mock_factory(self, mock_factory):
        """Тест с mock фабрикой"""
        mock_book = MagicMock()
        mock_factory.return_value = mock_book

        library = Library()
        document = library.add_document("book", "Any Title", "Any Author", 2023)

        mock_factory.assert_called_once_with("book", "Any Title", "Any Author", 2023)
        self.assertEqual(document, mock_book)

    def test_mock_adapter(self):
        """Тест с mock адаптером"""
        mock_adapter = MagicMock()
        mock_adapter.add_document.return_value = 42
```

```
        library = Library()
        library.adapter = mock_adapter

        book = Book("Test Book", "Author", 2023)
        library.documents.append(book)

        # Имитируем вызов через адаптер
        doc_id = library.adapter.add_document(book)
        self.assertEqual(doc_id, 42)
        mock_adapter.add_document.assert_called_once_with(book)


class TestMockIntegration(unittest.TestCase):
    """Интеграционные тесты с моками"""

    @patch('library_system.DocumentFactory.create_document')
    @patch('library_system.LibraryAdapter')
    def test_complete_workflow_with_mocks(self, MockAdapter, mock_factory):
        """Полный workflow с моками"""
        # Настройка моков
        mock_book = MagicMock()
        mock_book.title = "Mock Book"
        mock_book.is_available.return_value = True
        mock_book.borrow.return_value = True
        mock_book.return_doc.return_value = True

        mock_factory.return_value = mock_book

        mock_adapter_instance = MagicMock()
        MockAdapter.return_value = mock_adapter_instance

        # Создание системы
        library = Library()
        library.factory = mock_factory
        library.adapter = mock_adapter_instance

        mock_notifier = MagicMock()
        library.notifier = mock_notifier

        # Выполнение workflow
        document = library.add_document("book", "Mock Book", "Author", 2023)
        borrow_result = library.borrow_document("Mock Book", "user123")
        return_result = library.return_document("Mock Book", "user123")

        # Проверки
        self.assertEqual(document, mock_book)
        self.assertTrue(borrow_result)
        self.assertTrue(return_result)

        mock_factory.assert_called_once_with("book", "Mock Book", "Author", 2023)
        self.assertEqual(mock_notifier.notify.call_count, 3)
```

```python
if __name__ == '__main__':
    unittest.main()
```

3. Результаты работы программы

ДЕМОНСТРАЦИЯ БИБЛИОТЕЧНОЙ СИСТЕМЫ
==

1. СОЗДАНИЕ ДОКУМЕНТОВ ЧЕРЕЗ ФАБРИКУ:

Email to admin@library.ru: 2025-11-23 16:06:45.790618: DOCUMENT_ADDED - 'Python Programming'
SMS to +7-999-123-45-67: DOCUMENT_ADDED - Python Programming
LOG: 2025-11-23 16:06:45.790618: DOCUMENT_ADDED - 'Python Programming'
Email to admin@library.ru: 2025-11-23 16:06:45.791077: DOCUMENT_ADDED - 'Science Today'
SMS to +7-999-123-45-67: DOCUMENT_ADDED - Science Today
LOG: 2025-11-23 16:06:45.791077: DOCUMENT_ADDED - 'Science Today'
Email to admin@library.ru: 2025-11-23 16:06:45.791456: DOCUMENT_ADDED - 'AI Research'
SMS to +7-999-123-45-67: DOCUMENT_ADDED - AI Research
LOG: 2025-11-23 16:06:45.791456: DOCUMENT_ADDED - 'AI Research'
 Создана книга: Python Programming
 Создан журнал: Science Today
 Создана статья: AI Research

2. РАБОТА АДАПТЕРА:

 ID документа в старой системе: 4
 Информация: Python Programming (Book)
 Доступен: True

3. ВЫДАЧА И ВОЗВРАТ ДОКУМЕНТОВ:

 Выдача книги...
Email to admin@library.ru: 2025-11-23 16:06:45.793034: DOCUMENT_BORROWED - 'Python Programming' by user123
SMS to +7-999-123-45-67: DOCUMENT_BORROWED - Python Programming
LOG: 2025-11-23 16:06:45.793034: DOCUMENT_BORROWED - 'Python Programming' by user123
 Возврат книги...
Email to admin@library.ru: 2025-11-23 16:06:45.793468: DOCUMENT_RETURNED - 'Python Programming' by user123
SMS to +7-999-123-45-67: DOCUMENT_RETURNED - Python Programming
LOG: 2025-11-23 16:06:45.793468: DOCUMENT_RETURNED - 'Python Programming' by user123

4. СТАТИСТИКА:

 Всего документов: 3
 Доступно: 3
 Email уведомлений: 5
 SMS уведомлений: 5
 Логов: 5
==
 Демонстрация завершена успешно!