Идея проекта

Проект представляет собой систему управления учебными материалами, в которой преподаватели могут создавать различные виды образовательных ресурсов, такие как тесты, практические задания и лекции. Для унификации процесса создания материалов используется Фабричный метод, обеспечивающий гибкость и расширяемость системы.

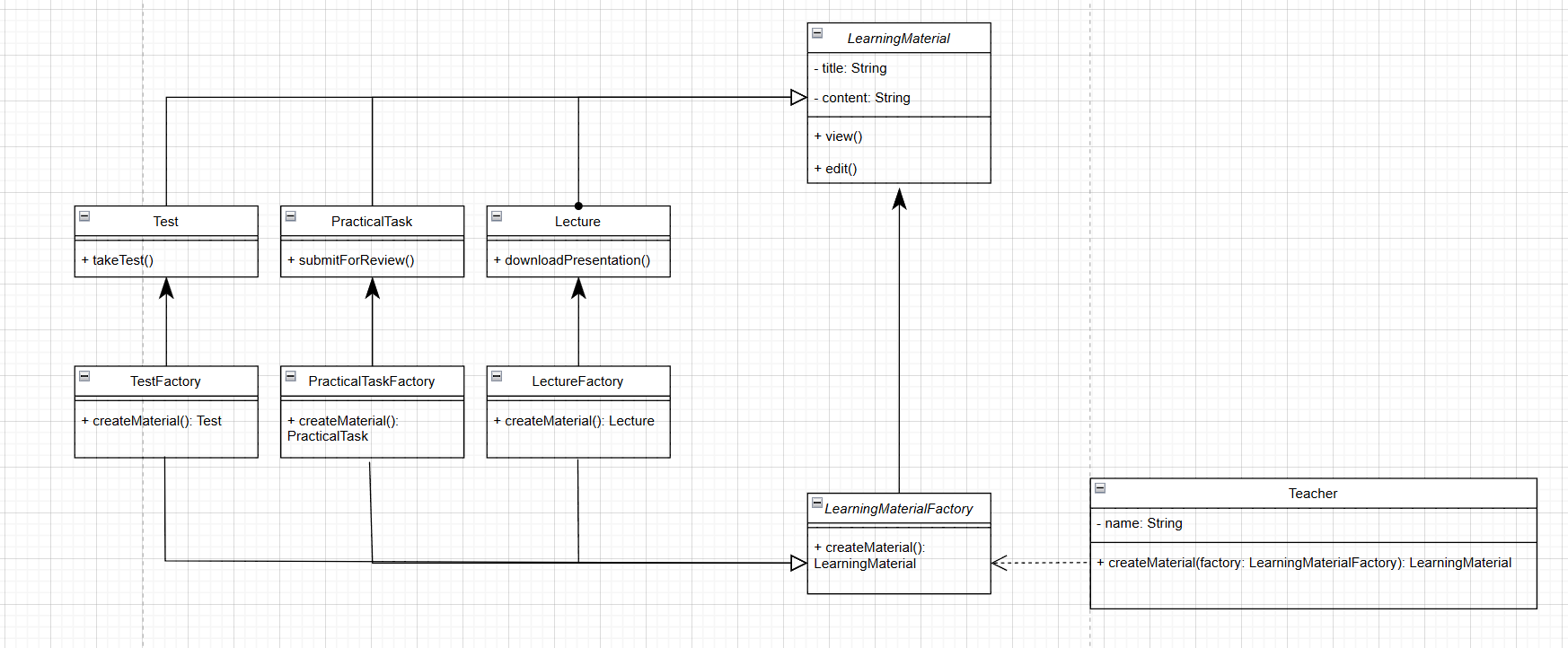
Описание проекта

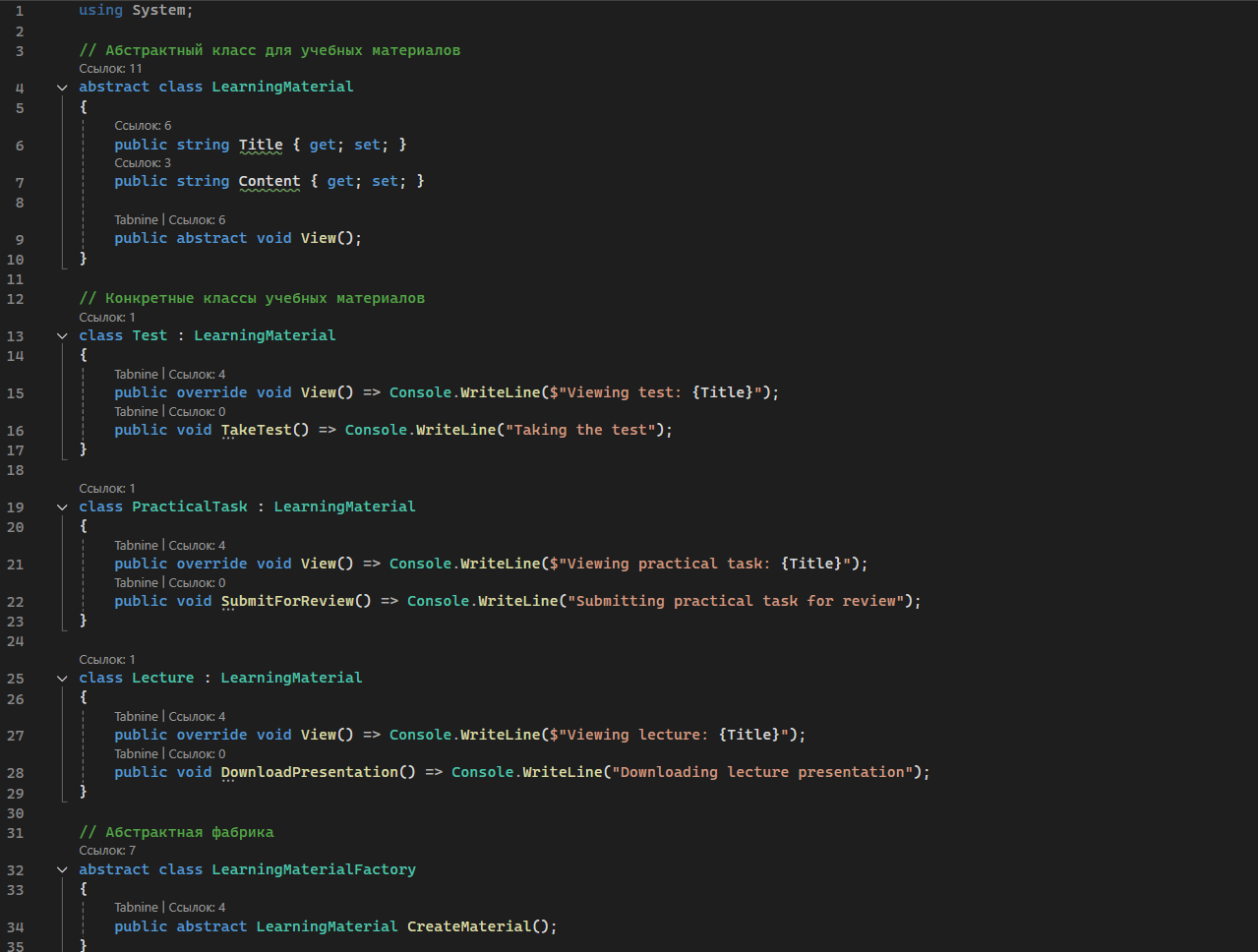
Система состоит из следующих ключевых компонентов:

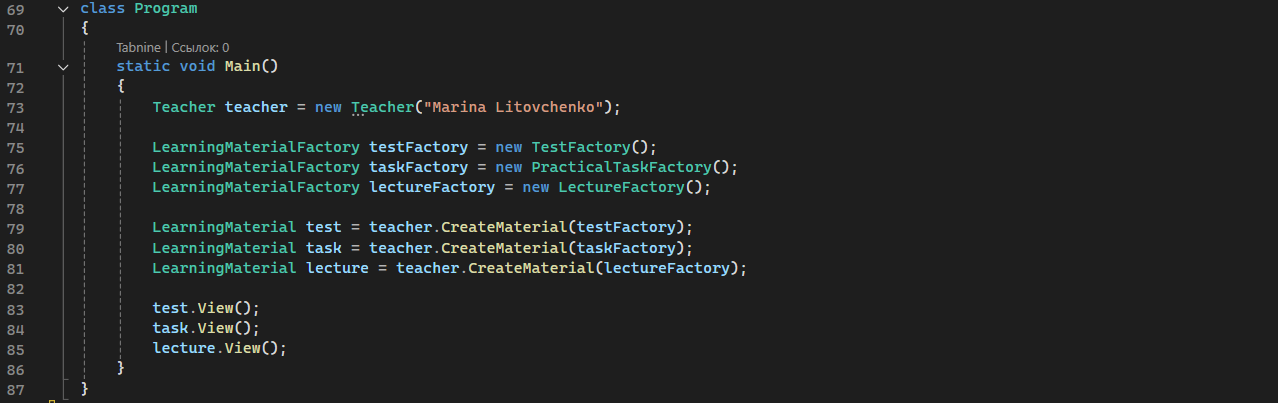
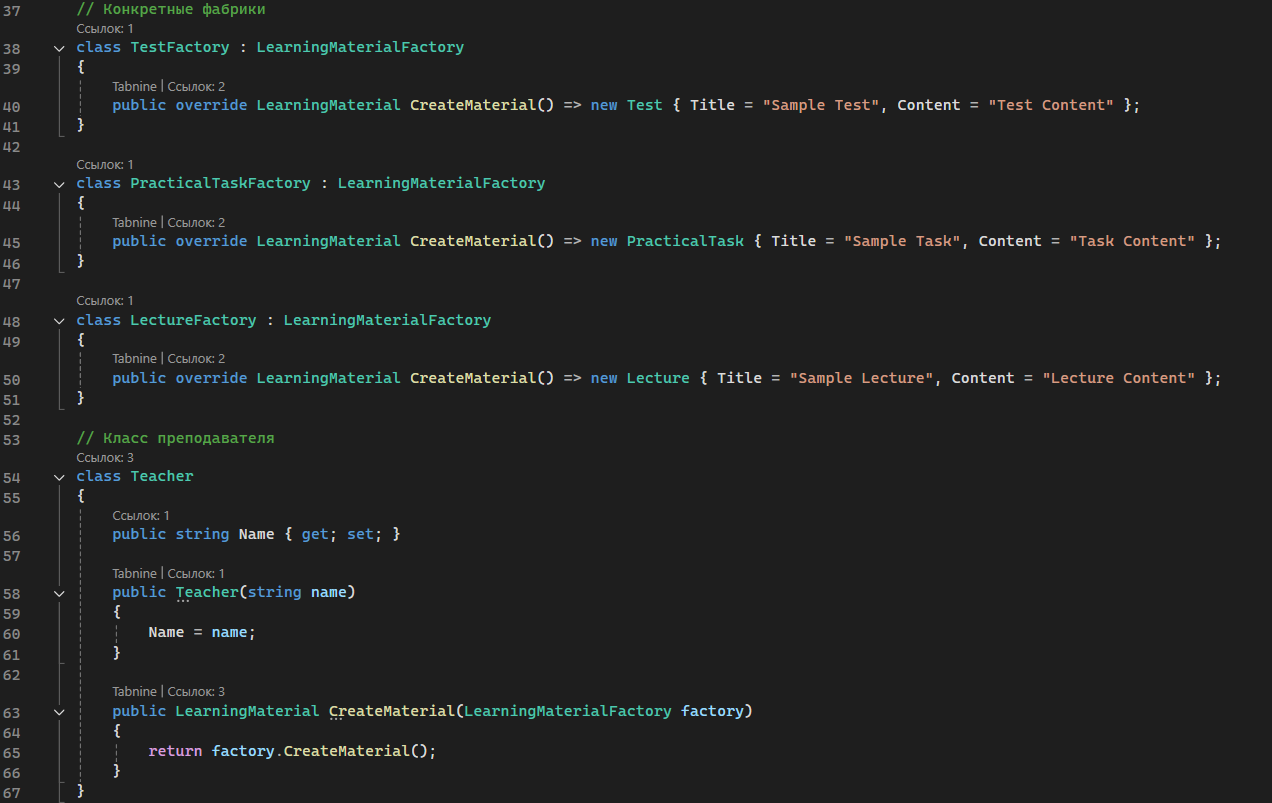
* LearningMaterial — базовый класс, представляющий учебный материал. Он хранит заголовок (title) и содержимое (content), а также предоставляет метод просмотра (view()).
* Test, PracticalTask, Lecture — классы, наследуемые от LearningMaterial. Каждый из них имеет уникальные методы:
  + Test — takeTest() для прохождения теста.
  + PracticalTask — submitForReview() для отправки на проверку.
  + Lecture — downloadPresentation() для скачивания лекции.
* Фабрики (TestFactory, PracticalTaskFactory, LectureFactory) — классы, реализующие создание соответствующих учебных материалов. Все они наследуются от LearningMaterialFactory, которая определяет общий метод createMaterial().
* Teacher — класс, представляющий преподавателя, который может использовать фабрику для создания новых учебных материалов.

Связи между классами

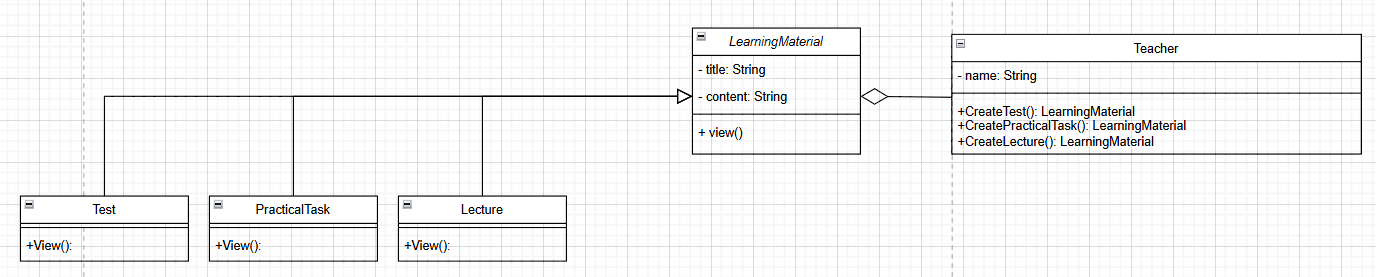
1. Test, PracticalTask и Lecture наследуются от LearningMaterial (обобщение).
2. TestFactory, PracticalTaskFactory и LectureFactory реализуют фабричный метод и ассоциированы с соответствующими классами материалов.
3. LearningMaterialFactory создаёт объекты LearningMaterial, но не наследуется от него.
4. Teacher использует LearningMaterialFactory, но не владеет ей (зависимость).



Код на языке C#:  


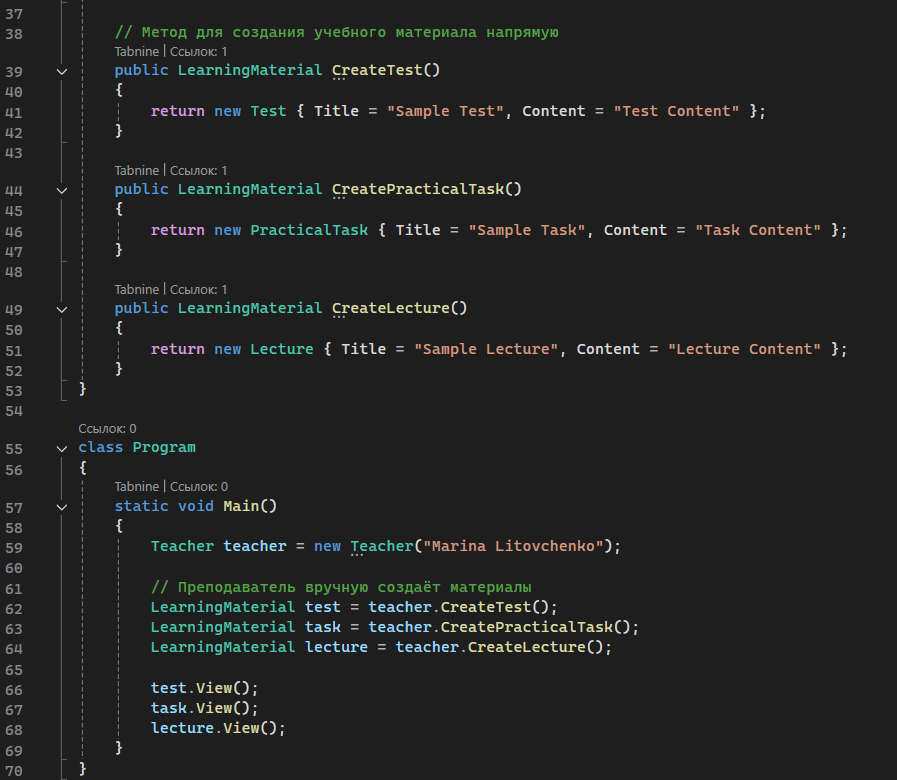


Решение задачи без использования паттерна:



Реализация на языке C#:





Что изменилось по сравнению с фабричным методом?

1. Преподаватель теперь сам создаёт объекты (new Test(), new PracticalTask(), new Lecture()).
   * Нет больше фабрик, метод CreateMaterial() отсутствует.
   * Вместо этого у Teacher появились три отдельных метода (CreateTest(), CreatePracticalTask(), CreateLecture()).
2. Нет абстрактной фабрики LearningMaterialFactory.
   * Вариант без фабричного метода прямо привязан к конкретным классам (Test, PracticalTask, Lecture).
   * Если появится новый тип материала (VideoLecture), придётся менять код класса Teacher, добавляя новый метод CreateVideoLecture().
3. Принципы ООП нарушены (особенно открытости/закрытости).
   * Код менее гибкий: если нужно поменять способ создания материалов, придётся менять код Teacher.
   * При использовании фабричного метода новый тип материала добавляется просто созданием новой фабрики, без изменений в Teacher.

|  |  |  |
| --- | --- | --- |
| Критерий | С фабричным методом | Без фабричного метода |
| Гибкость | Легко добавить новый тип | Нужно менять код преподавателя |
| Инкапсуляция | Процесс создания материалов скрыт | Teacher знает, какие классы создавать |
| Поддерживаемость | Легко расширять | Труднее расширять |
| Простота кода | Чуть сложнее (но понятнее) | Простейший код, но жёстко привязан к классам |

Вывод:

* В простых проектах можно использовать второй вариант (без фабрик).
* В расширяемых системах фабричный метод лучше, потому что позволяет легко добавлять новые типы материалов без изменений в основном коде.