Лабораторная работа 1

Компоненты:

CustomDataTable:

Визуальный компонент вывода таблицы значений

Публичные Методы:

- Определить структуру таблицы

```
void ConfigureColumns(params CustomDataTableColumnParameter[]
columnParameters)
```

CustomDataTableColumnParameter

```
/// <summary>
/// Параметры столбца таблицы
/// </summary>
public record CustomDataTableColumnParameter
{
  /// <summary>
  /// Заголовок
  /// </summary>
  public string HeaderName { get; init; } = string.Empty;

  /// <summary>
  /// Ширина
  /// </summary>
  public int Width { get; init; } = 0;

  /// <summary>
  /// Видимость
  /// </summary>
  public bool Visible { get; init; } = true;

  /// <summary>
  /// Название свойства
  /// </summary>
  public string PropertyName { get; init; } = string.Empty;
}
```

- Отчистить таблицу

```
public void Clear()
```

- Заполнить таблицу

```
public void Fill<TType>(IList<TType> insertValues)
```

Публичные Функции:

- Получить стоку таблицы в виде объекта

```
public TType? GetRow<TType>(int rowIndex) where TType : new()
```

Свойства:

- Индекс выбранной строки

```
public int SelectedRow
```

CustomListBox

Визуальный компонент выбора из списка значений

Публичные Методы:

- Заполнить список значениями

```
public void FillValues(IEnumerable<string> strings)
```

- Очистить список

```
public void Clear()
```

Свойства:

- Событие возникающие при изменении списка

```
public event EventHandler ValueChanged
```

- Обработка ошибок

```
public event Action<Exception> AnErrorOccurred
```

- Выбранное значение

```
public string Selected
```

CustomNumericInputField

Визуальный компонент ввода целочисленного значения допускающего null

Свойства:

- Событие возникающие при изменении поля ввода

```
public event EventHandler NumericInputChanged
```

- Обработка ошибок

```
public event Action<Exception> AnErrorOccurred
```

- Значения поля ввода

```
public int? Value
```

  Возможно исключение `InvalidNumericInputValueException`

Лабораторная работа 2

Компоненты:

CustomPdfTable

Компонент для сохранения таблицы в пдф

Публичные Методы:

- Определить структуру таблицы

```
public void SaveToPdf(PdfTableInfo tableInfo)
```

PdfTableInfo

```
/// <summary>
/// Параметры для создания таблиц в пдф
/// </summary>
public record PdfTableInfo
{
  /// <summary>
  /// имя файла (включая путь до файла)
  /// </summary>
  public string FilePath { get; init; } = @"C:\pdfTable.pdf";

  /// <summary>
  /// название документа(заголовок в документе)
  /// </summary>
  public string Title { get; init; } = "Таблица";

  /// <summary>
  /// Список таблиц
  /// </summary>
  public IEnumerable<string[,]> Tables { get; init; } = [];
}
```

CustomPdfTableWithGrouping

Компонент создающий таблицу и группирует элементы по 1 столбцу

Публичные Методы:

- Сохранить в пдф

```
public void SaveToPdf<TType>(PdfTableWithGroupingInfo<TType> tableInfo) where
TType : class
```

PdfTableWithGroupingInfo

```
/// <summary>
/// Параметры для создания таблицы в пдф с группировкой по 1 столбцу
/// </summary>
public class PdfTableWithGroupingInfo<TType> where TType : class
{
/// <summary>
/// имя файла (включая путь до файла)
```

```
    /// </summary>
    public string FilePath { get; init; } = @"C:\pdfTable.pdf";

    /// <summary>
    /// название документа(заголовок в документе)
    /// </summary>
    public string Title { get; init; } = "Таблица";

    /// <summary>
    /// Высота заголовков
    /// </summary>
    public float HeaderHeight { get; init; } = 0.5f;

    /// <summary>
    /// Параметры столбцов
    /// </summary>
    public IEnumerable<ColumnInfo> Columns { get; init; } = [];

    /// <summary>
    /// Список таблиц
    /// </summary>
    public IEnumerable<RowInfo<TType>> Rows { get; init; } = [];
    }
```

CustomPdfHistogram

Компонент создающий линейную диаграмму

Публичные Методы:

- Сохранить гистограмму в пдф

```
    public void SaveToPdf(PdfHistigramInfo histogramInfo)
```

PdfHistigramInfo

```
    /// <summary>
    /// Параметры для создания линейной диаграммы
    /// </summary>
    public record PdfHistigramInfo
    {
        /// <summary>
        /// Имя файла (включая путь до файла)
        /// </summary>
        public string FilePath { get; init; } = @"C:\pdfTable.pdf";

        /// <summary>
        /// Заголовок документа
        /// </summary>
        public string DocumentTitle { get; init; } = "Гистограмма";

        /// <summary>
        /// Заголовок диаграммы
        /// </summary>
        public string HistogramTitle { get; init; } = "Гистограмма";

        /// <summary>
```

```
        /// Расположение легенды
        /// </summary>
        public PdfLegendPosition LegendPosition { get; init; } =
    PdfLegendPosition.Bottom;

        /// <summary>
        /// Значения
        /// </summary>
        public required IEnumerable<PdfHistogramLineInfo> Values { get; init; }
    }
```

Лабораторная работа 3

Программный продукт предназначен для учета успеваемости студентов и управления их данными.
Он позволяет хранить информацию о каждом студенте,
формировать отчеты в различных форматах (Word, PDF, Excel)
и визуализировать данные с помощью круговой диаграммы.

Цель программы

Цель программы — автоматизация процесса учета успеваемости студентов, упрощение работы с данными и формирование отчетов для анализа успеваемости и стипендиальных выплат.

Функционал программы

- Хранение данных о студентах:

1. ФИО студента.
2. Дата поступления
3. Форма обучения (выбирается из выпадающего списка).
4. Успеваемость

- Отображение данных в виде списка
  Данные о студентах выводятся на главной форме в виде списка строк
- Формирование документа в PDF:
  Создается таблица с данными о средних баллах студентов по сессиям.
  Колонки таблицы: сессии от 1 до 6.
  Строки таблицы: средние баллы студентов (без указания ФИО).
- Формирование таблицы в excel:
  Создается табличный отчет с информацией по всем студентам.

  Шапка таблицы:

  - Id
  - ФИО
  - Образование
    - Форма обучения

- Дата поступления

- Создание круговой диаграммы в Word:
  Формируется линейная диаграмма, отображающая распределение студентов по форме обучения,
  поступивших в разнык года

Компонент	Преимущества	Недостатки
CustomDateTimePicker	Понятные и логичные названия переменных	Занимаемое компонентом место не оптимизировано (могут возникнуть трудности с контейнерами)
VisualSelectionComponent	Содержит большую часть требуемого функцинонала	В базовой поставке отсутствует функционал заполнения, отсутствует xml документация, функции имеют не оправдано длинное название, границы компонента занимают чуть больше места чем вложенные элементы
ComponentDiagram	Реализует функционал создания диограммы в текстовом документе word	Использует комерческую библиотеку с бесплатным доступом, на документе остаётся водяной знак, в функцию создания документа передаётся объект, но его поля нужно инициализировать в конструкторе, отсутствует xml документация
CustomExcelTable	Удобные настройки таблицы, понятные наименования функций и классов	Отсутствует xml документация

Лабораторная работа №5.

Применение структурных паттернов.

Задание

1) Дать описание паттернов, указанных во вариантах, для каких целей они могут применяться,
   какие участники там фигурируют.

2) На основе задания из 3 лабораторной работы, для каждого паттерна придумать сущности, относящиеся к той же предметной области, что описаны в задании и реализация которых бы в приложении потребовала применения паттерна.

Ограничения:

- На каждый паттерн свои сущности

- В качестве источника сущностей использовать предметную область задания 3 лабораторной работы, а не элементы разработки (что-то типа «У меня паттерн Singleton, укажу ка я класс-подключение к БД через него», не принимается).

3) Создать диаграммы классов, отражающие взаимодействие новых сущностей (а также используемый паттерн) с классами, созданными в рамках 3 лабораторной работы. Отдельно отметить классы, которые являются участниками паттерна

Вариант 3: *Adapter*, *Composite*, *Proxy*

Паттерн Adapter

Паттерн Адаптер используется для того,
чтобы объекты с несовместимыми интерфейсами могли работать вместе.
Он оборачивает один интерфейс в другой,
делая несовместимые классы совместимыми без изменения их исходного кода.

Задачи, которые решает паттерн Adapter:

- Интеграция стороннего кода или библиотек:
  когда требуется использовать стороннюю библиотеку с вашим приложением,
  но её интерфейс отличается от имеющегося интерфейса.
- Обратная совместимость: когда нужно подключить новый код к старому интерфейсу, не изменяя существующий.
- Упрощение взаимодействия: уменьшает сложность работы с несколькими несовместимыми компонентами.

Участники:

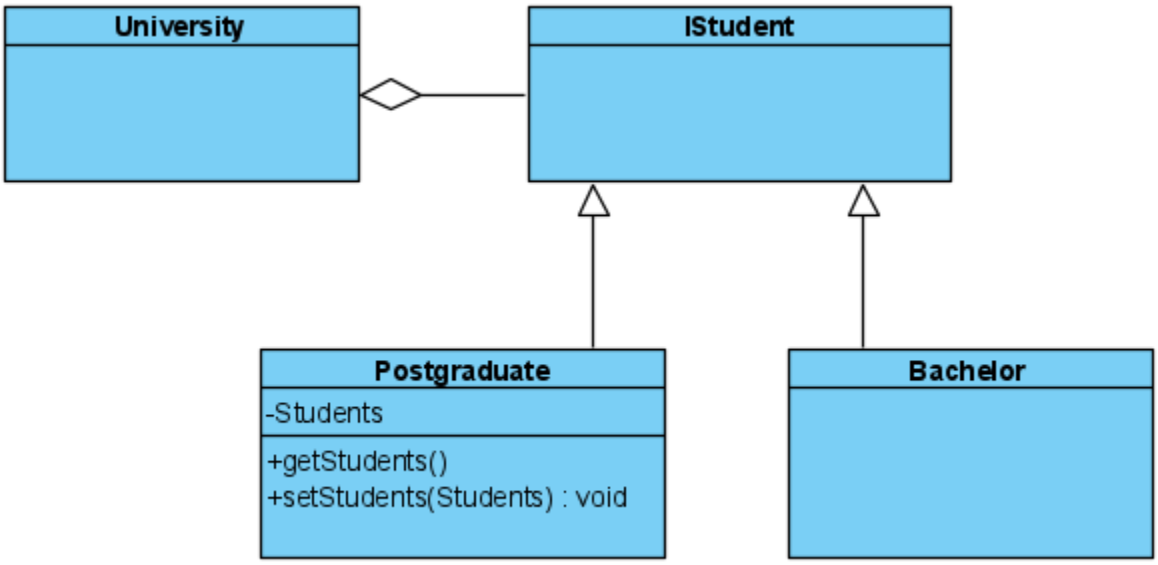
- Target: представляет объекты, которые используются клиентом.
- Client: использует объекты Target для реализации своих задач.
- Adaptee: представляет адаптируемый класс, который хотелось бы использовать у клиента вместо объектов Target.
- Adapter: сам адаптер, который позволяет работать с объектами Adaptee как с объектами Target.

Пример реализации для рассматриваемой предметной области:

Классы, реализующие паттерн:

- IStudentTracker (Target) – Интерфейс системы отслеживания успеваемости студентов
- StudentManager (Client) – Использует данные о успеваемости, для определеня судьбы студентов
- AttendanceTracker (Adaptee) – Реализализовывает систему отслеживания посещаемости студентов

- StudentTrackerAdapter (Adapter) – "оборачивает" стороннюю систему и преобразует её в интерфейс, ожидаемый клиентом



Паттерн Composite

Паттерн Компоновщик (Composite) объединяет группы объектов в древовидную структуру по принципу "часть-целое и позволяет клиенту одинаково работать как с отдельными объектами, так и с группой объектов.

Образно реализацию паттерна можно представить в виде меню,
которое имеет различные пункты. Эти пункты могут содержать подменю, в которых,
в свою очередь, также имеются пункты. То есть пункт меню служит с одной стороны частью меню,
а с другой стороны еще одним меню. В итоге мы однообразно можем работать как с пунктом меню,
так и со всем меню в целом.

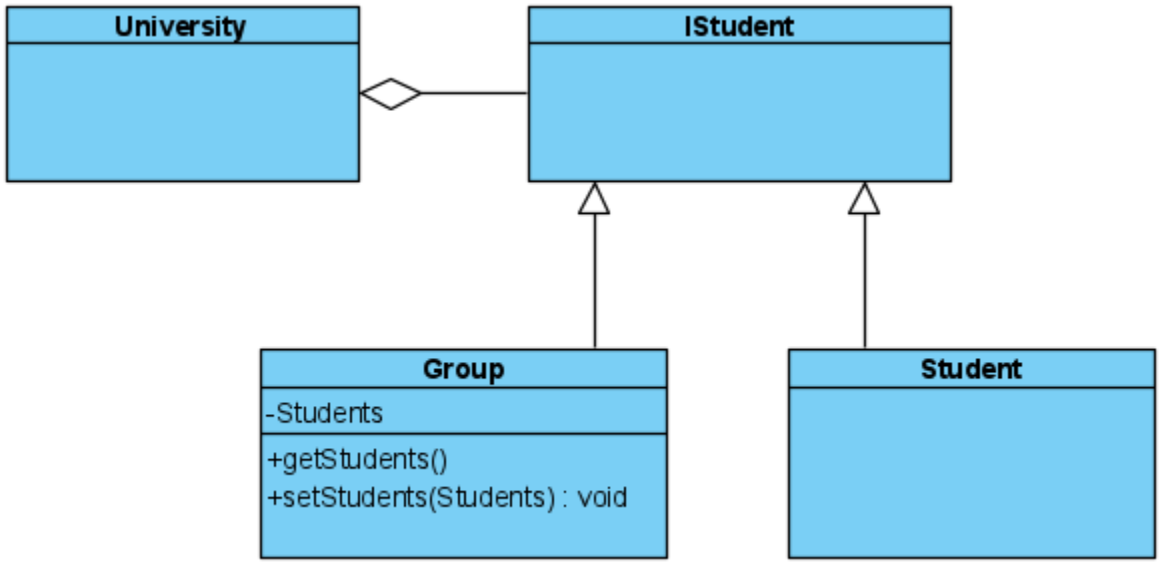
Участники:

- Component – определяет интерфейс для всех компонентов в древовидной структуре
- Composite – представляет компонент, который может содержать другие компоненты и реализует механизм для их добавления и удаления
- Leaf – представляет отдельный компонент, который не может содержать другие компоненты
- Client – клиент, который использует компоненты

Классы, реализующие паттерн:

- IStudent (Component) – Студент обучающийся в вузе
- Group (Composite) – Группировка студентов (групаа в потоке, поток, курс...)
- Student (Leaf) – Конкретный студент

- University (Client) – Обеспечивает обучение студентов



Паттерн Proxy

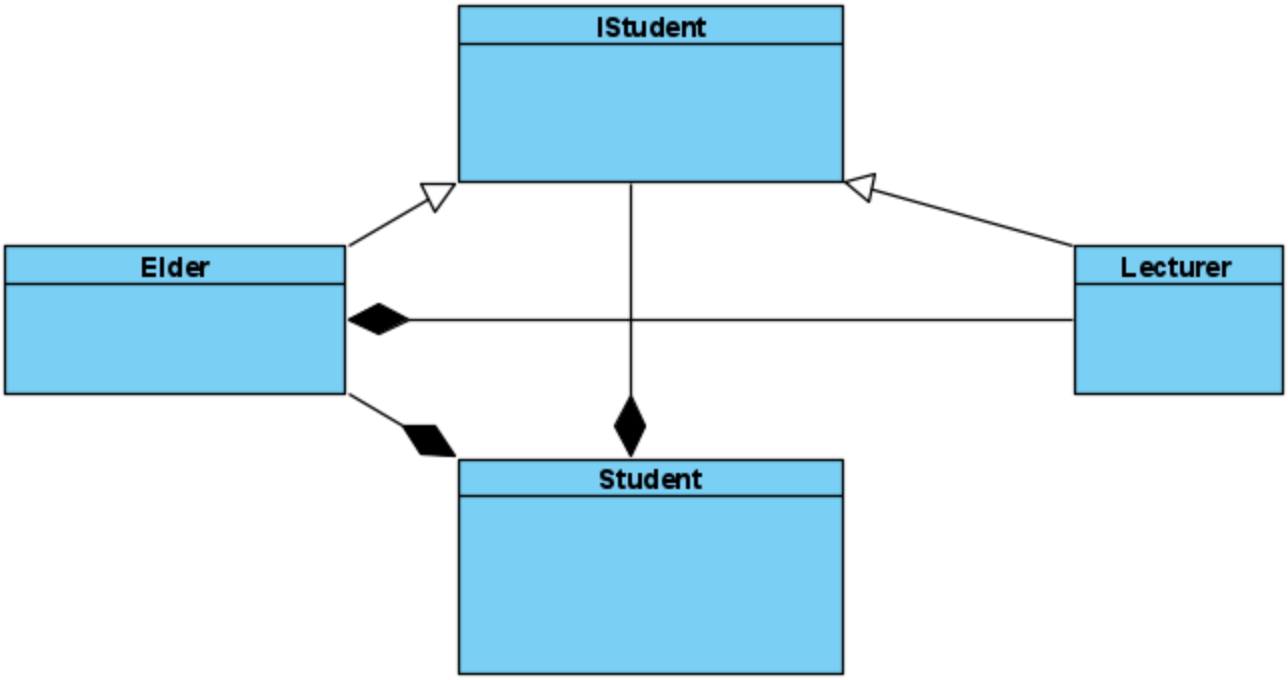
Паттерн Заместитель (Proxy) предоставляет объект-заместитель, который управляет доступом к другому объекту.
То есть создается объект-суррогат, который может выступать в роли другого объекта и замещать его.

Участники:

- Subject – определяет общий интерфейс для Proxy и RealSubject. Поэтому Proxy может использоваться вместо RealSubject
- RealSubject – представляет реальный объект, для которого создается прокси
- Proxy – заместитель реального объекта. Хранит ссылку на реальный объект, контролирует к нему доступ,
  может управлять его созданием и удалением. При необходимости Proxy переадресует запросы объекту RealSubject
- Client – использует объект Proxy для доступа к объекту RealSubject

Классы, реализующие паттерн:

- IStudent (Subject) – Студент
- Student (RealSubject) – Студент из группы
- Elder (Proxy) – Староста отвечает на вопросы преподователя, при надомности спрашивает студентов группы
- Lecturer (Client) - Задаёт вопросы студентам через старосту



Лабораторная работа №6.

Применение структурных паттернов.

Задание

1) Дать описание паттернов, указанных во вариантах, для каких целей они могут применяться,
какие участники там фигурируют.

2) На основе задания из 3 лабораторной работы, для каждого паттерна придумать сущности, относящиеся к той же предметной области, что описаны в задании и реализация которых бы в приложении потребовала применения паттерна.

Ограничения

- На каждый паттерн свои сущности
- В качестве источника сущностей использовать предметную область задания 3 лабораторной работы,
а не элементы разработки (что-то типа «У меня паттерн Singleton,
укажу ка я класс-подключение к БД через него», не принимается).

3) Создать диаграммы классов, отражающие взаимодействие новых сущностей (а также используемый паттерн) с классами, созданными в рамках 3 лабораторной работы. Отдельно отметить классы, которые являются участниками паттерна

Вариант 3 *AbstractFactory, FactoryMethod, ObjectPool.*

Паттерн AbstractFactory

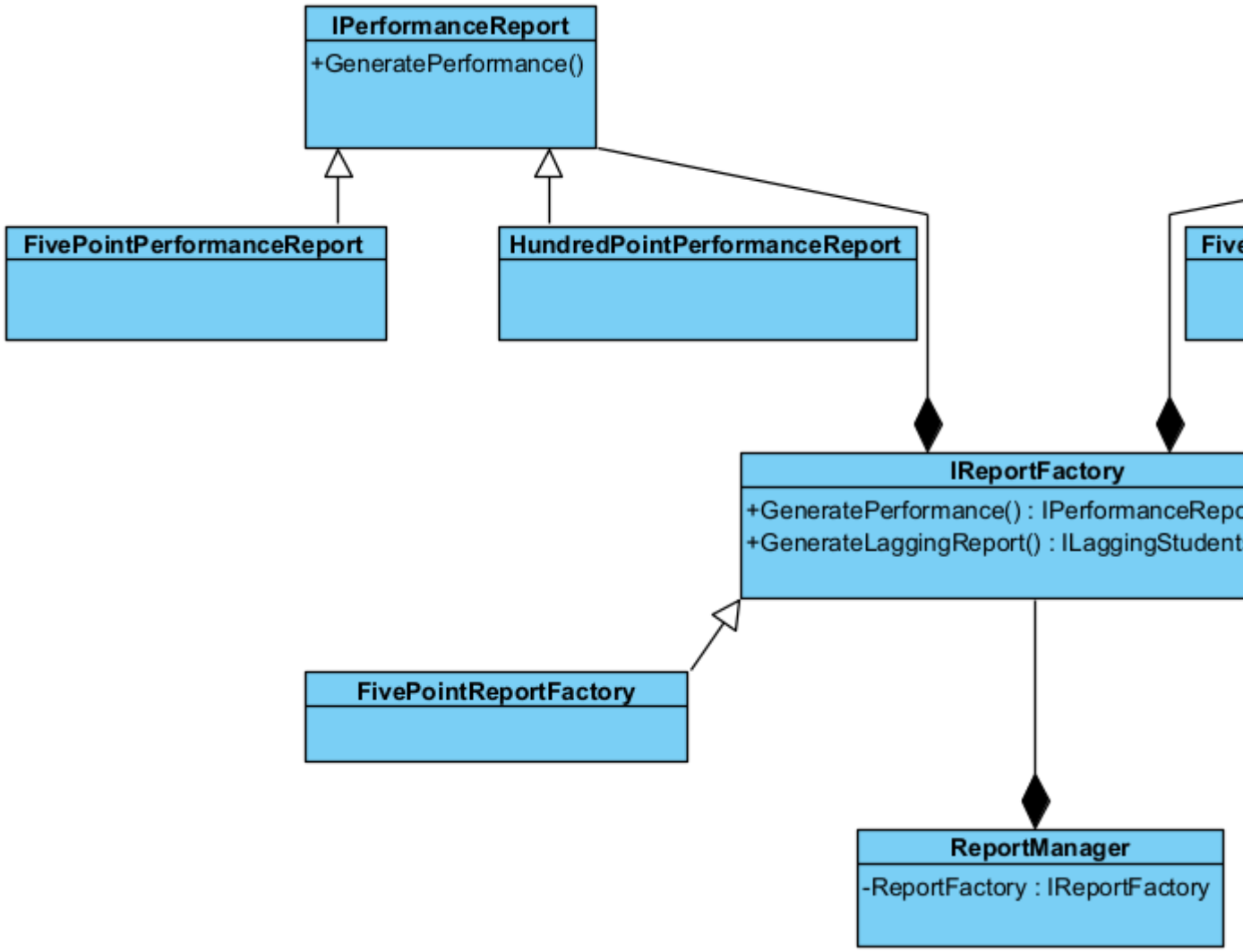
Паттерн "Абстрактная фабрика" (Abstract Factory) предоставляет интерфейс для создания семейств взаимосвязанных объектов с определенными интерфейсами без указания конкретных типов данных объектов.

Участники

- AbstractProductA и AbstractProductB – абстрактные классы, определяющие интерфейс для классов,
  объекты которых будут создаваться в программе
- ProductA1 / ProductA2 и ProductB1 / ProductB2 – конкретные классы,
  представляющие конкретную реализацию абстрактных классов.
- AbstractFactory - Абстрактный класс фабрики определяет методы для создания объектов.
  Причем методы возвращают абстрактные продукты, а не их конкретные реализации.
- Client – класс клиента, использующий класс фабрики для создания объектов.
  При этом он использует исключительно абстрактный класс фабрики AbstractFactory
  и абстрактные классы продуктов AbstractProductA и AbstractProductB и никак не зависит от их конкретных реализаций.

Классы, реализующие паттерн:

- IReportFactory (Abstract Factory) интерфейс фабрики для создания отчетов.
- FivePointReportFactory (Concrete Factory) фабрика для 5-балльной системы.
- HundredPointReportFactory (Concrete Factory) фабрика для 100-балльной системы.
- IPerformanceReport (Abstract Product) интерфейс для отчета по успеваемости.
- FivePointPerformanceReport и HundredPointPerformanceReport (Concrete Products) конкретные реализации отчетов по успеваемости для 5- и 10-балльной систем.
- ILaggingStudentsReport (Abstract Product) интерфейс для отчета по отстающим студентам.
- FivePointLaggingStudentsReport и HundredPointLaggingStudentsReport (Concrete Products) отчеты по отстающим студентам для 5- и 10-балльной систем.
- ReportManager (Client) класс, который обрабатывает отчеты



Паттерн FactoryMethod

Фабричный метод (Factory Method) - это паттерн, который определяет интерфейс для создания объектов некоторого класса,
но непосредственное решение о том, объект какого класса создавать происходит в подклассах.
То есть паттерн предполагает, что базовый класс делегирует создание объектов классам-наследникам.

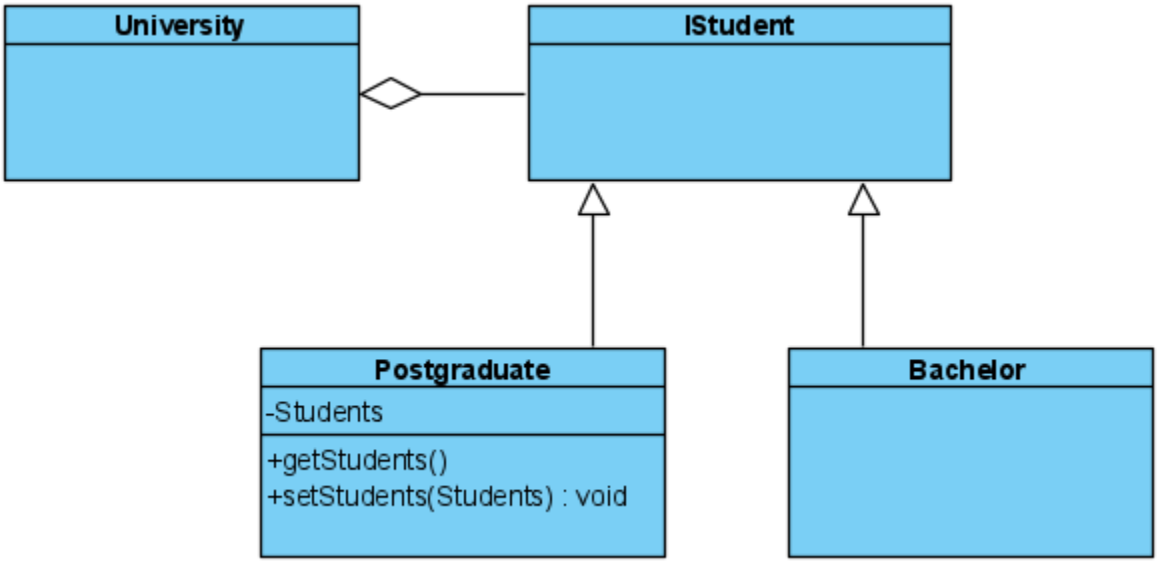
Участники:

- Product – интерфейс или абстрактный класс, определяющий структуру создаваемых объектов.
- ConcreteProductA и ConcreteProductB – конкретные реализации интерфейса “Product”, представляющие первый и второй тип продукта.
- Creator – абстрактный класс или интерфейс, который определяет фабричный метод для создания объектов типа “Product”.

- ConcreteCreatorA и ConcreteCreatorB – конкретные реализации “Creator”, которая создаёт объекты типа “ConcreteProductA” и “ConcreteCreatorB” соответственно.

Классы, реализующие паттерн:

- Product IReport – определяет интерфейс для всех отчетов.
- ConcreteProductA и ConcreteProductB GradeReport и AttendanceReport – конкретные реализации интерфейса “IReport”.
- Creator ReportCreator – определяет фабричный метод для создания объектов типа “IReport”.
- ConcreteCreatorA и ConcreteCreatorB GradeReportCreator и AttendanceReportCreator – конкретные реализации “ReportCreator”.



Паттерн ObjectPool

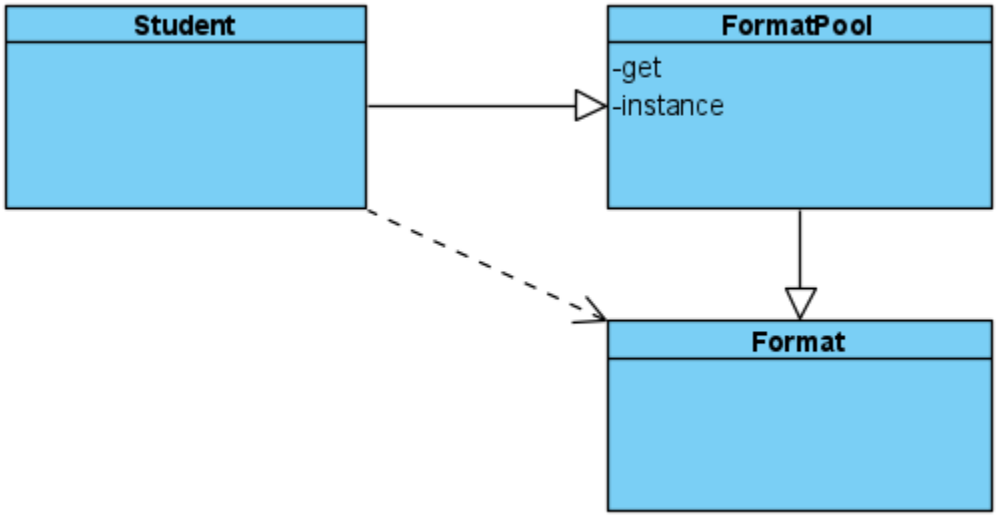
Object Pool (пул объектов) — это поведенческий паттерн, который используется для оптимизации управления ресурсами.
Он позволяет многократно использовать уже созданные объекты вместо их повторного создания и уничтожения.
Это особенно полезно, когда создание объектов является дорогой операцией с точки зрения времени или ресурсов.

Участники:

- Pool – управляет коллекцией объектов. Отвечает за выдачу и возврат объектов из пула. Решает, когда создавать новый объект или использовать существующий.
- ReusableObject (повторно используемый объект) - объект, который находится в пуле и может быть многократно использован. Включает логику и состояние объекта.
- Client - получает объект из пула для использования. Возвращает объект обратно в пул, когда он больше не нужен.

Классы, реализующие паттерн:

- Pool FormatPool – управляет доступностью объектов “Format”. Позволяет выдавать объекты формы обучения для студентов.
- ReusableObject Format – содержит данные, связанные с конкретной формой обучения.
- Client Student – представляет студента и содержит данные о нем.



Лабораторная работа №7.

Применение структурных паттернов.

Задание

1) Дать описание паттернов, указанных во вариантах, для каких целей они могут применяться,
   какие участники там фигурируют.
2) На основе задания из 3 лабораторной работы, для каждого паттерна придумать сущности, относящиеся к той же предметной области, что описаны в задании и реализация которых бы в приложении потребовала применения паттерна.

Ограничения

- На каждый паттерн свои сущности
- В качестве источника сущностей использовать предметную область задания 3 лабораторной работы,
  а не элементы разработки (что-то типа «У меня паттерн Singleton,
  укажу ка я класс-подключение к БД через него», не принимается).

3) Создать диаграммы классов, отражающие взаимодействие новых сущностей (а также используемый паттерн) с классами,

созданными в рамках 3 лабораторной работы. Отдельно отметить
классы, которые являются участниками паттерна

Вариант 3 *Mediator, Mediator (2 различных реализации), Strategy.*

Паттерн Mediator

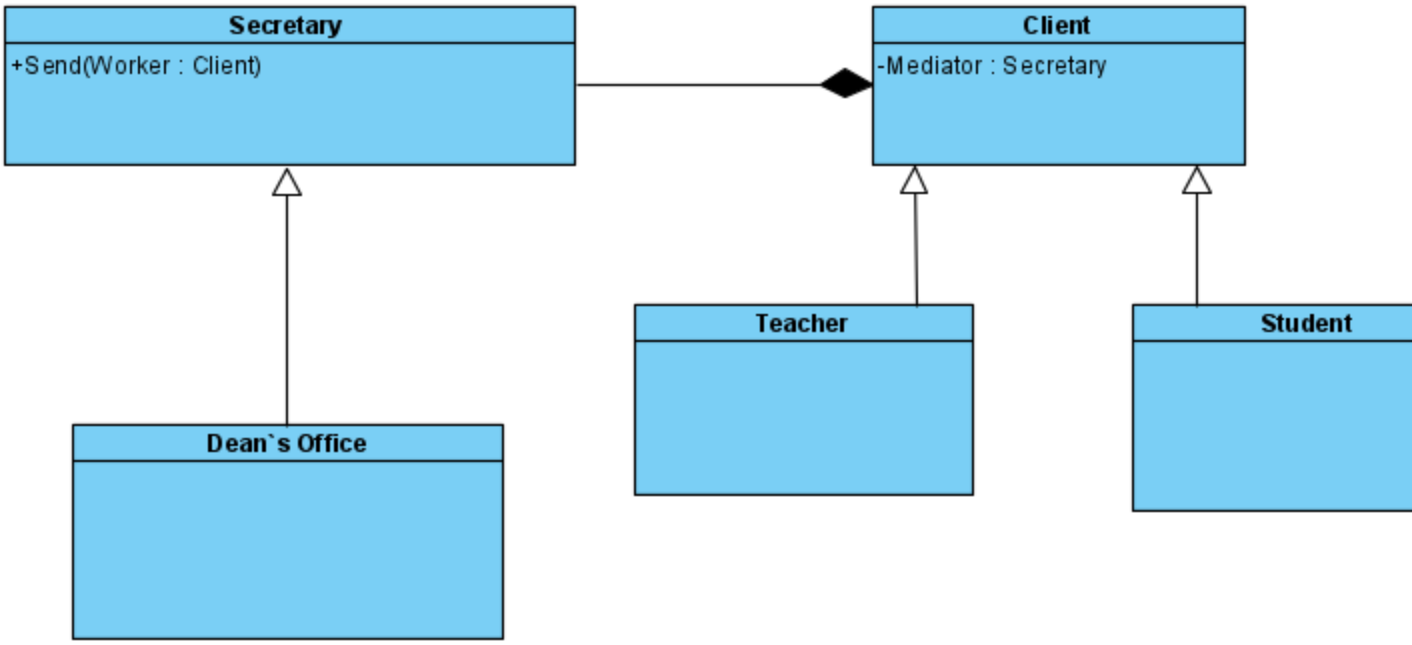
Паттерн "Посредник" (Mediator) представляет такой шаблон проектирования,
который обеспечивает взаимодействие множества объектов без необходимости ссылаться друг на друга.
Тем самым достигается слабосвязанность взаимодействующих объектов.

Участники

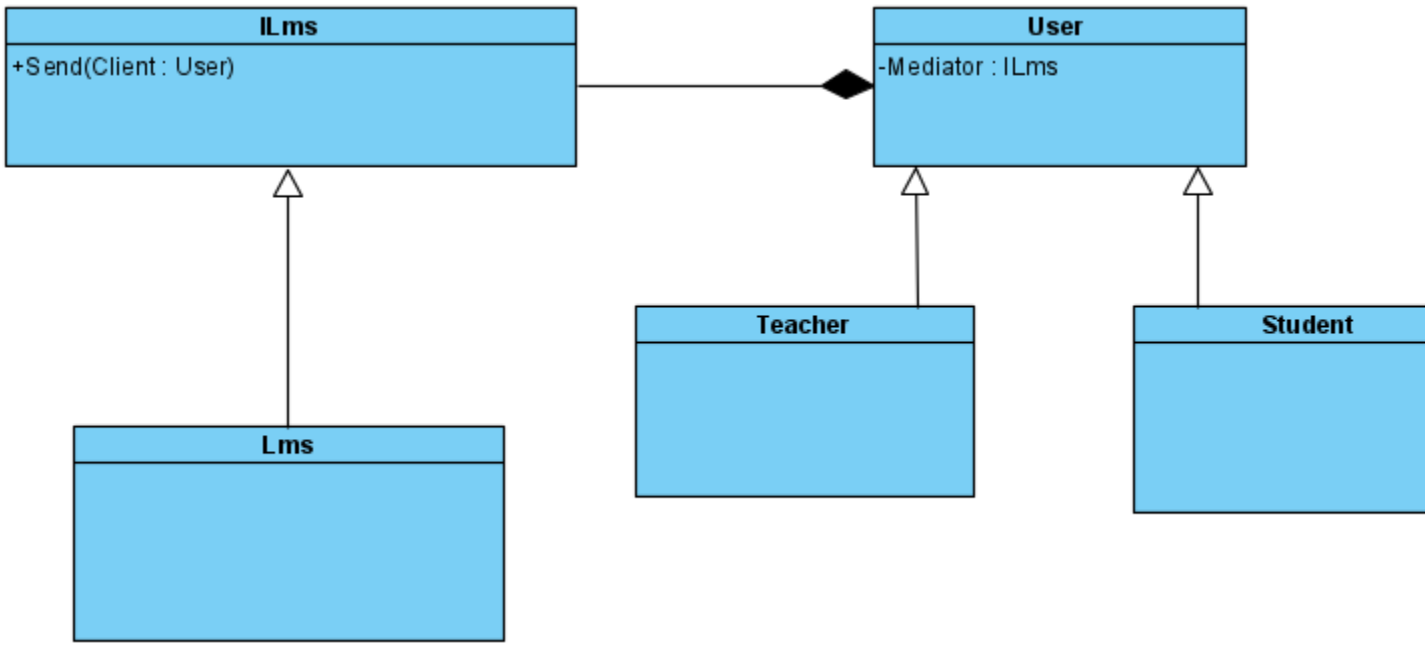
- Mediator - представляет интерфейс для взаимодействия с объектами Colleague.
- Colleague - представляет интерфейс для взаимодействия с объектом Mediator.
- ConcreteColleague1 и ConcreteColleague2 - конкретные классы коллег,
  которые обмениваются друг с другом через объект Mediator.
- ConcreteMediator - конкретный посредник, реализующий интерфейс типа Mediator.

Классы, реализующие паттерн:

- Secretary Mediator - Секретарь
- Client Colleague - Участник вузовской системы
- Teacher ConcreteColleague1 - Преподователь
- Student ConcreteColleague2 - Студент
- Dean`s Office ConcreteMediator - Деканат



- ILms Mediator - Сайт лмс
- User Colleague - Пользователь лмс
- Teacher ConcreteColleague1 - Преподователь
- Student ConcreteColleague2 - Студент
- Lms ConcreteMediator - Логика сайта лмс



Паттерн Strategy

Strategy (Стратегия) — шаблон проектирования, который определяет набор алгоритмов, инкапсулирует каждый из них и обеспечивает их взаимозаменяемость. В зависимости от ситуации мы можем легко заменить один используемый алгоритм другим. При этом замена алгоритма происходит независимо от объекта, который использует данный алгоритм.

Участники:

- IStrategy: интерфейс, который определяет метод Algorithm(). Это общий интерфейс для всех реализующих его алгоритмов. Вместо интерфейса здесь также можно было бы использовать абстрактный класс.
- ConcreteStrategy1 и ConcreteStrategy2: классы, которые реализуют интерфейс IStrategy, предоставляя свою версию метода Algorithm(). Подобных классов-реализаций может быть множество.
- Context: класс, который хранит ссылку на объект IStrategy и связан с интерфейсом IStrategy отношением агрегации.

Классы, реализующие паттерн:

- IStrategy IReportStrategy – общий интерфейс для всех стратегий формирования отчетов.
- PdfReportStrategy, ExcelkReportStrategy, WordReportStrategy - классы, реализующие формирование отчета в pdf, excel и word.
- Context ReportContext – контекст, который использует стратегию для формирования отчета.

Этот подход позволяет легко добавлять новые форматы отчетов (например, JSON или HTML) без изменения существующего кода, реализуя принцип открытости/закрытости.