Старовойтова Александра гр.932020

Вариант 2.

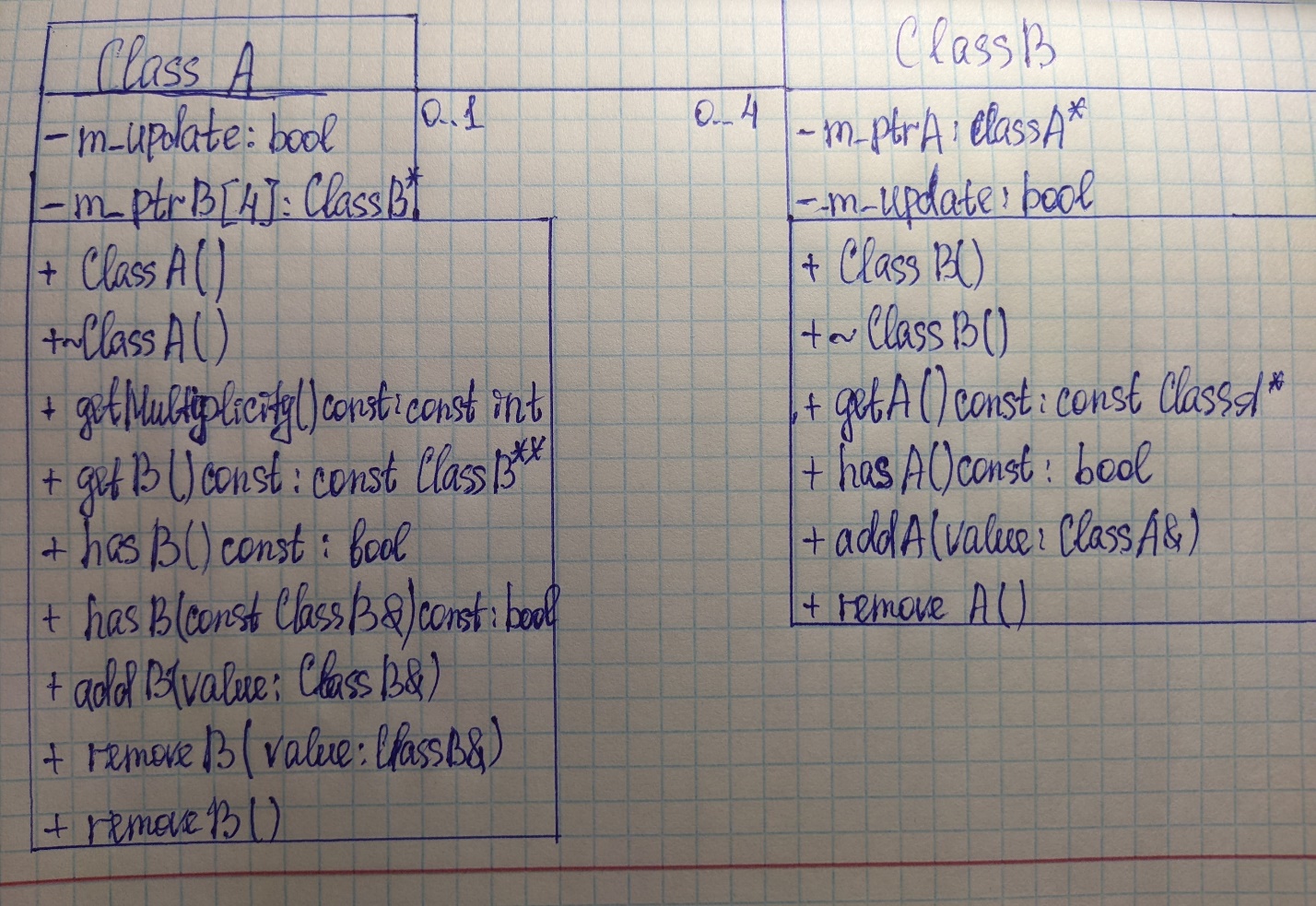
Задание 1.

Класс ClassA имеет член-массив указателей на связанные объекты ClassB. Это означает, что один объект ClassA может иметь несколько связей с объектами ClassB. Класс ClassA предоставляет методы для работы с этими связями, такие как добавление, удаление и проверка наличия связей. Класс ClassB имеет указатель на связанный объект ClassA. Это означает, что каждый объект ClassB может быть связан только с одним объектом ClassA. Класс ClassB также предоставляет методы для установки, удаления и проверки наличия связи с объектом ClassA.

То есть, каждый объект класса ClassA может быть связан с несколькими объектами класса ClassB, но каждый объект класса ClassB может быть связан только с одним объектом класса ClassA. Это определяет отношение "один-ко-многим" между классами ClassA и ClassB.

Вывод: между классами ClassA и ClassB существует двунаправленная ассоциация,

* Класс ClassA может иметь от 0 до 4 связей с объектами класса ClassB.
* Класс ClassB может иметь только одну связь с объектом класса ClassA.



Задание 2.

Тест ответа для удобства продублирован в коде по ссылке: <https://onlinegdb.com/3HRb-qJrFB>

В представленном коде есть нарушение принципа открытости/закрытости (Open/Closed Principle). Принцип открытости/закрытости гласит, что программные сущности (классы, модули, функции и т.д.) должны быть открыты для расширения, но закрыты для модификации. Он помогает создавать гибкую архитектуру, которая может быть легко расширена новым функционалом без необходимости изменения существующего кода.

В данном случае, класс DrwManager содержит жестко закодированные условия для рисования фигур. Когда добавляется новый тип фигуры (например, треугольник), необходимо изменять код внутри метода drawShapes, добавлять новое условие и соответствующую логику рисования. Это нарушает принцип открытости/закрытости, так как при добавлении нового типа фигуры нужно модифицировать существующий код.

Для исправления этой проблемы можно применить принцип полиморфизма и использовать виртуальные функции. Вместо жестко закодированных условий для каждого типа фигуры, можно определить виртуальную функцию draw в базовом классе Shape, которую каждый дочерний класс будет переопределять по-своему. Затем, в цикле drawShapes можно вызывать эту виртуальную функцию для каждого элемента в списке shapeList, и правильная реализация draw будет автоматически выбрана в соответствии с типом объекта.

В исправленном коде добавлен виртуальный деструктор в базовый класс Shape, чтобы обеспечить правильное удаление объектов при наследовании. Класс DrwManager теперь использует список указателей на базовый класс Shape с помощью std::shared\_ptr, что позволяет автоматически управлять памятью объектов. В методе drawShapes вызывается полиморфная функция Draw, которая рисует каждую фигуру соответствующим образом, и не требует изменений при добавлении новых типов фигур.

Задание 3.

<https://onlinegdb.com/we5tTHaT4A>

Паттерн "Абстрактная фабрика" позволяет создавать семейства взаимосвязанных объектов без указания их конкретных классов. Он предоставляет интерфейс для создания объектов каждого типа в семействе, при этом клиентский код не зависит от конкретных классов создаваемых объектов, а работает с абстрактными интерфейсами.

В контексте предложенной легенды паттерн "Абстрактная фабрика" может быть использован для создания фабрик, которые будут отвечать за создание объектов смартфонов и простых телефонов для каждого производителя.

В данном примере мы создаем иерархию классов для телефонов, включая интерфейс Phone и абстрактные продукты Smartphone (смартфон) и BasicPhone (простой телефон).

Конкретные продукты, которые образуют семейство (типы телефонов).

Затем определяем абстрактную фабрику PhoneFactory, которая содержит методы для создания смартфона и простого телефона.

Затем создаем конкретные фабрики для каждого производителя телефонов: NokiaFactory, SamsungFactory и HTCFactory. Каждая из этих фабрик реализует методы createSmartphone и createBasicPhone, которые создают соответствующие конкретные продукты - телефоны соответствующих производителей.

В основном цикле main мы используем оператор switch case для создания фабрики и соответствующих продуктов-телефонов для каждого производителя. Затем выводим информацию о созданных продуктах, включая их имена, с использованием функции getName().

Простите за плохую схему, я не успела ее нормально нарисовать, не рассчитала, что программа получится настолько большой. 