Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**ИСПОЛЬЗОВАНИЕ И ПРИЁМЫ РАЗРАБОТКИ КЛАССОВ**

Отчёт по лабораторной работе №4 по дисциплине

«Объектно-ориентированное программирование»

Студент группы 588-1

\_\_\_\_\_\_\_\_\_\_ / Чан Хыу Тхай

«\_» \_\_\_\_\_\_\_\_\_2022 г.

Руководитель старший научный

сотрудник, доцент каф. КСУП

\_\_\_\_\_\_\_\_\_ / Горяинов А.Е.

«\_\_\_» 2022 г.

Томск 2022

1. **Цель работы**

Целью данной лабораторной работы является использование и разработка классов, а также приобретение практических навыков решения актуальных задач объектно-ориентированного программирования.

1. **Теоретические основы**

**Класс** – это просто независимая подпрограмма, в которой есть свой набор переменных и функций (обязательно изучи урок про функции). В основной программе мы можем создать экземпляр класса (он же называется объект) и пользоваться теми инструментами, которые имеются в классе. Использование классов позволяет:

* Разделить сложную программу на отдельные независимые части
* Создавать удобные библиотеки
* Использовать свои “наработки” в другом проекте, не переписывая один и тот же код
* Облегчить и упростить программу, если в ней используются повторяющиеся конструкции и алгоритмы

Классы очень похожи на структуры (читай урок по структурам), как по объявлению, так и по использованию, но класс является гораздо более мощной единицей языка благодаря механизмам наследования и прочим ООП-штукам.

Класс объявляется при помощи ключевого слова class и содержит внутри себя члены класса – переменные и функции:

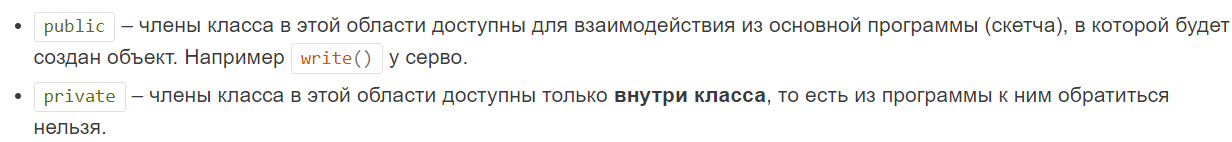
class Имя\_класса {

член1;

член2;

};

Важное отличие от структуры: содержимое класса делится на области: публичные и приватные. Они определяются при помощи ключевых слов public и private, область действует до начала следующей области или до закрывающей фигурной скобки класса:



class Имя\_класса {

public:

// список членов, доступных в программе

private:

// список членов для использования внутри класса

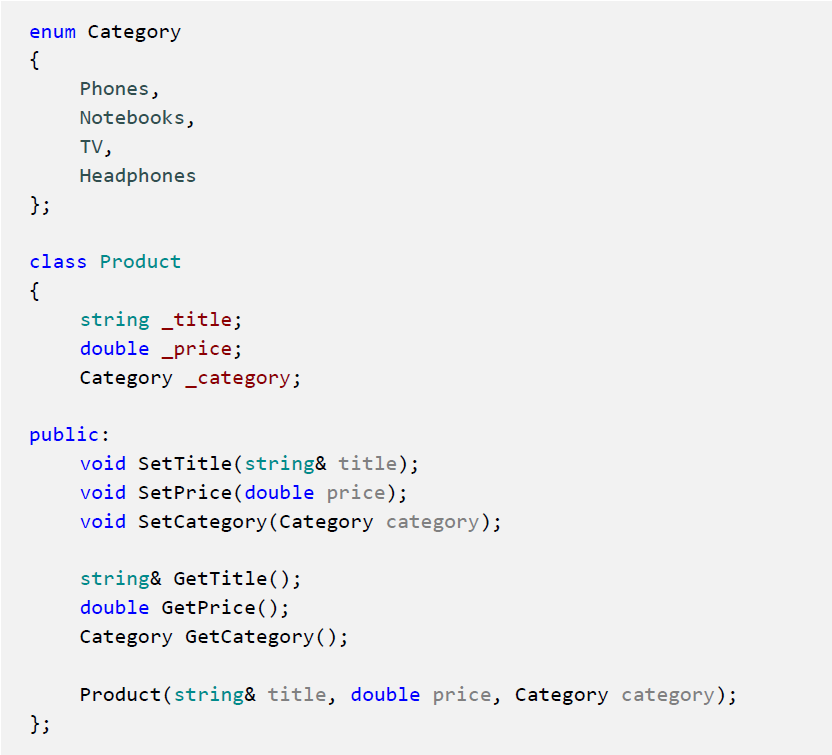
};

Ранее было рассмотрено агрегирование – взаимодействие двух классов, где объекты одного класса явля-ются частью состояния другого класса. Связь агрегирования предполагает, что один объект предметной об-ласти является частью другого, например, двигатель как часть автомобиля, или песня как часть музыкаль-ного альбома. Однако в реальном мире объекты не обязательно связаны агрегированием.

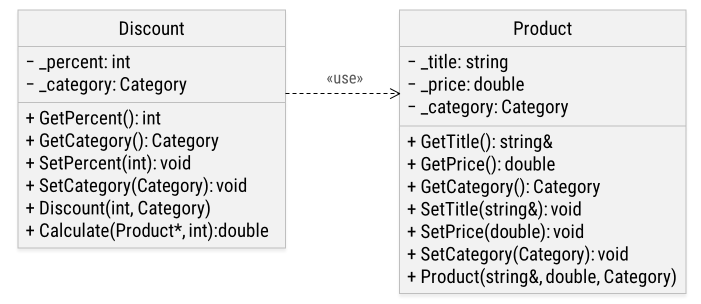
Например, покупатель и продавец в магазине взаимодействуют для совершения покупки, но ни в коем случае нельзя сказать, что покупатель является частью продавца или продавец – частью покупателя. Фактически, покупатель – это объект, который существует в среде самостоятельно, однако время от времени, для совер-шения действия (которое является частью его поведения) ему нужен объект продавца. Покупатель обраща-ется к продавцу, передавая деньги и запрос на товары, а продавец принимает деньги и возвращает запро-шенные товары. После завершения действия покупатель никак не связан с объектом продавца. Такой вид взаимодействия называется использованием.

**Использование** – взаимодействие двух объектов, при котором один объект обращается к полям или методам другого объекта во время выполнения собственных методов. Рассмотрим на более конкретном примере.

Допустим у нас есть класс Товара в магазине, у которого есть название, цена и категория (перечисление):

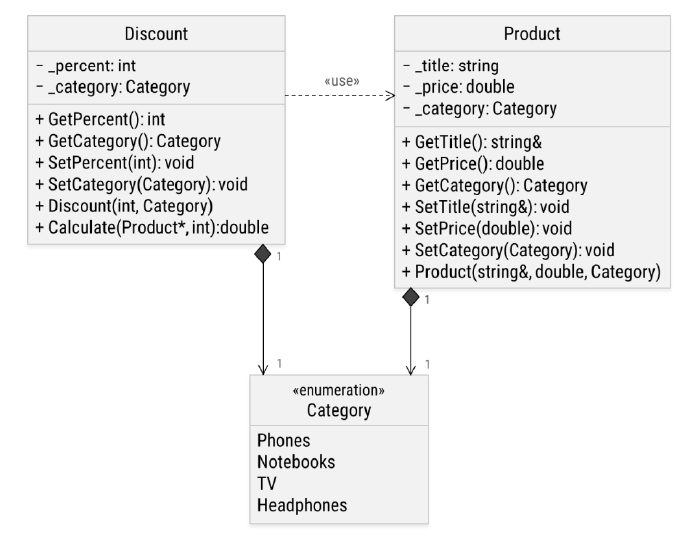


На UML-диаграммах классов использованием обозначается направленной пунктирной стрелкой со стереоти-пом (подписью) «use». Стрелка направляется от использующего класса к используемому:



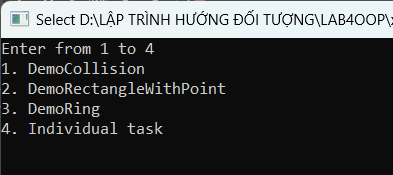
Для связи использования кратность не указывается. Отдельно стоит пояснить, что если два класса связаны агрегированием, то, логично, они связаны и использованием (например, потому что тип данных используе-мого класса явно указан в сеттерах и геттерах). Однако, если между двумя классами существует более одной связи, то на диаграммах указывается более сильная связь. То есть из двух связей агрегирования или исполь-зования на диаграмме рисуют только агрегирование.

Полная диаграмма примера будет выглядеть следующим образом:



1. **Ход работы**

Ниже представлен интерфейс лабораторной работы:



В первой задаче необходимо исправить разработанные классы, если класс агрегирует массив значений, массив и количество элементов массива задаются двумя отдельными сеттерами, переделайте отдельные сеттеры массива в один сеттер для указать зависимые поля. Это должны быть классы Album и Band. (рис. 1), (рис. 2).

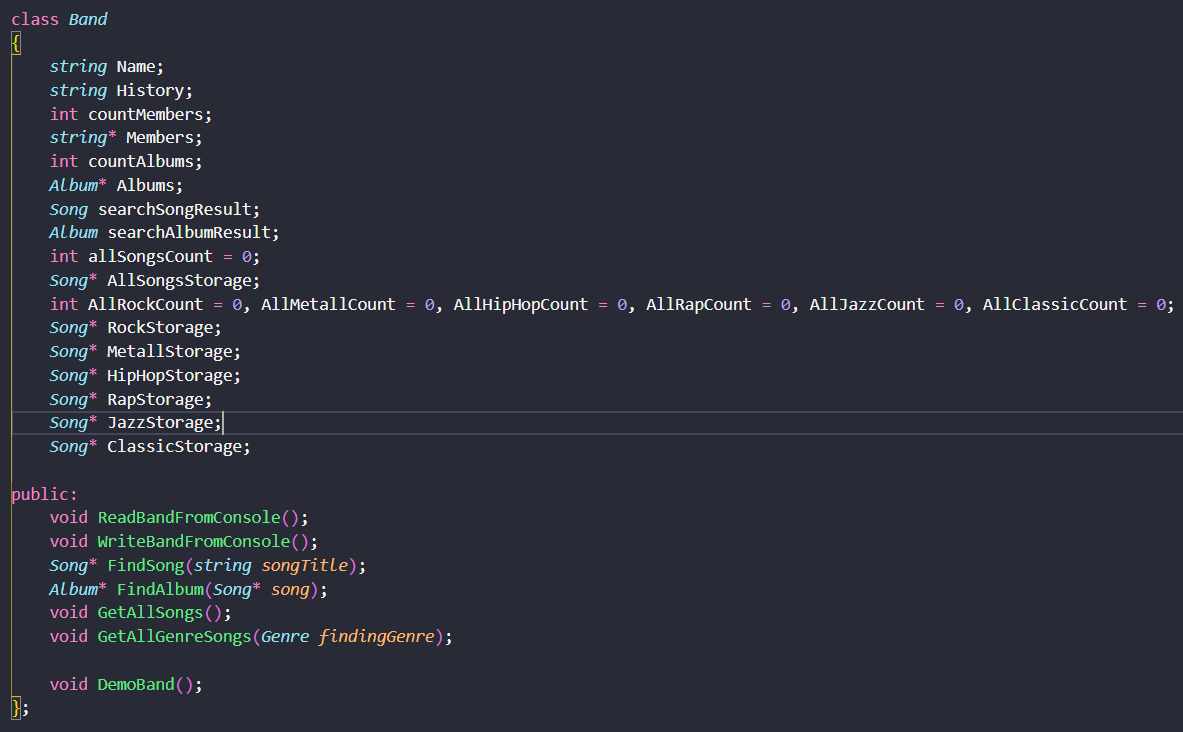


Рис. 1 – Создать класс Band

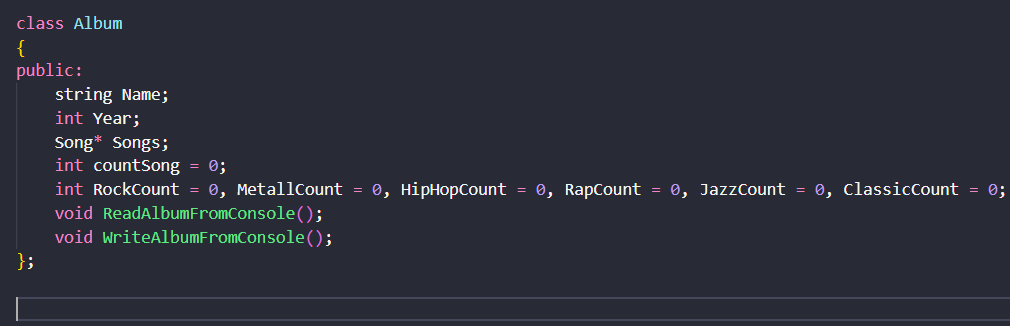


Рис. 2 – Создать класс Album

Следующей задачей необходимо создать статический класс CollisionManager - класс, выполняющий проверку о пересечении/столк-новении геометрических фигур. Классы по проверке столкновений применяются в геометрических САПР, 2D- и 3D-редакторах и компьютерных играх. Класс должен реализовать два метода:

- bool IsCollision(Rectangle&, Rectangle&) – метод принимает два экземпляра прямоугольника и проверяет не пересекаются ли они. Прямоугольники считаются пересекающимися если разница их координат по X (по модулю) меньше суммы половин их ширин и разница их координат по Y (по модулю) меньше суммы половин их высот. Если прямоугольники пересекаются, метод возвращает true, иначе возвращает false. (рис. 3).



Рис. 3 – Создать класс CollisionManager

Следующей задачей необходимо создать сервисный (не статический) класс GeometricProgram, в который поместите все демонстра-ционные функции, связанные с геометрическими фигурами: DemoRing(), DemoCollision() и другие ра-нее разработанные функции. Класс GeometricProgram является чистой выдумкой пользовательского интерфейса для лабораторной работы №4. Реализация пользовательского интерфейса в виде клас-сов также часто встречается на практике (рис. 4), (рис. 5).

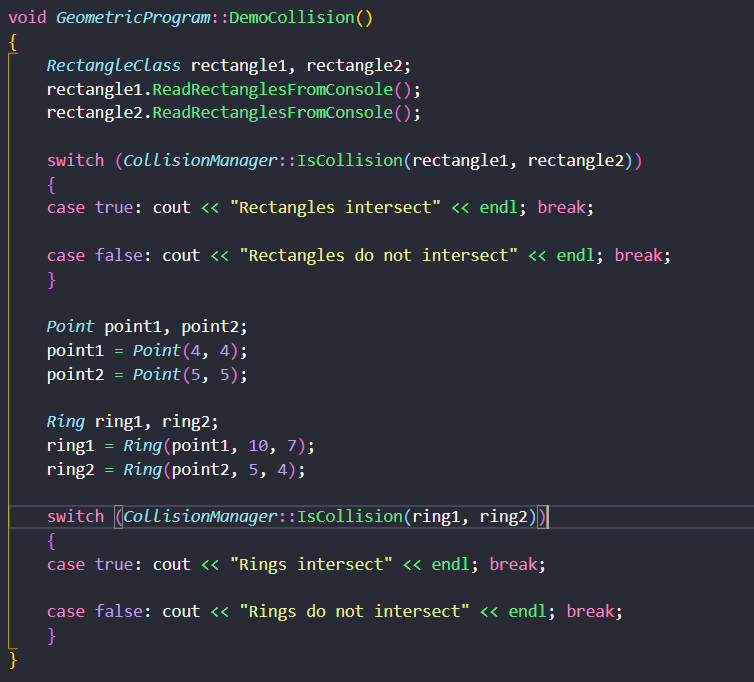


Рис. 4 – Создать функцию DemoCollision

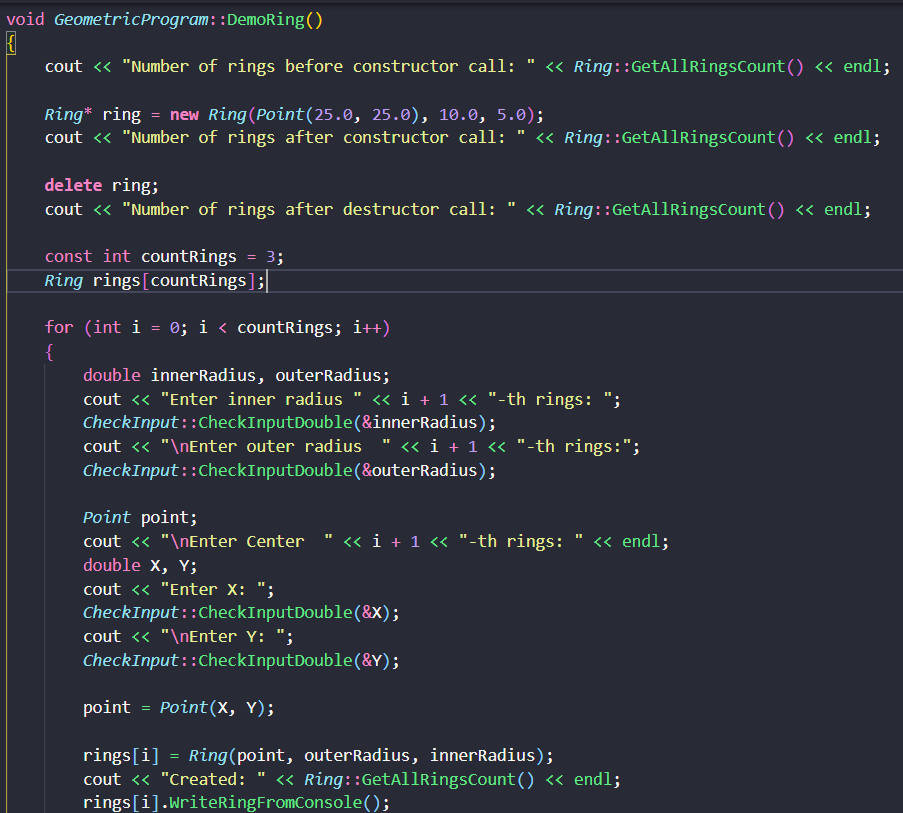


Рис. 5 – Создать функцию DemoRing

На рисунке ниже показана функция DemoCollision (рис. 6).

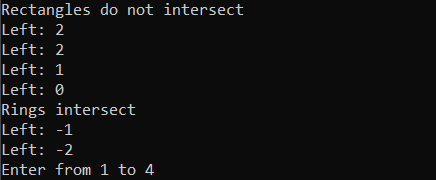


Рис. 6 – Результат демонстрационной функции

На рисунке ниже показана функция DemoRectangleWithPoint (рис. 7).

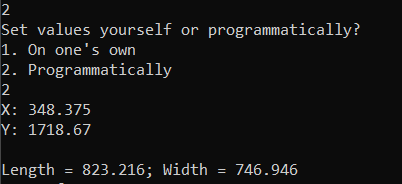


Рис. 7 – Результат демонстрационной функции

На рисунке ниже показана функция DemoRectangleWithPoint (рис. 8).

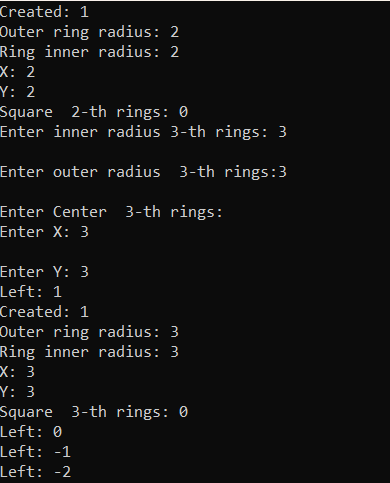


Рис. 8 – Результат демонстрационной функции

1. **Вывод**

В ходе работы в лаборатории были выполнены все задачи, разработана природа «Использование и приёмы разработки классов» в объектно-ориентированном программировании и набор функций для работы с ней.