Данная тетрадь содержит отношения между классами - определения и примеры реализации на C++

1. Зависимость

Зависимость - показывает, что один класс зависит от другого, чтобы выполнить свою функциональность (другими словами, при изменении независимого класса, зависимы от него может поменять своё поведение).

Это самая слабая связь между классами. Наиболее распространённый пример - класс X использует в своём методе метод класса Y, то есть X зависит от Y.

```
class DependencyEngine {
public:
        void start() {
                cout << "Engine started!" << endl;
        }
};

class DependencyCar {
public:
        void start(DependencyEngine& engine) {
                engine.start();
                cout << "Car started!" << endl;
        }
};
```

Объяснение: В этом примере класс DependencyCar зависит от класса DependencyEngine, так как для запуска автомобиля требуется запустить двигатель.

Метод start класса DependencyCar принимает объект DependencyEngine в качестве параметра.

2. Ассоциация

Ассоциация - показывает, что объекты одного класса связаны с объектами другого класса.

Существует три частных случая ассоциации - Агрегация, Композиция, Обобщение.

2.1 Агрегация

Агрегация - это отношение "часть-целое", где объекты составляют часть объекта-контейнера, но не зависят от него.

То есть объект-контейнер не управляет их временем жизни (есил контейнер будет уничтожен, то его содержимое - нет).

```
class Student {
public:
        string m_name;

        Student(const string& name) : m_name(name) {}
};

class University {
public:
        vector<Student*> students;
        void addStudent(Student* student) {
                students.push_back(student);
        }
        void displayStudents() {
                for (const auto& student : students) {
                        cout << student->m_name << endl;
                }
        }
};
```

Объяснение: В этом примере University содержит объекты Student, но Student не зависит от University.

Если University будет уничтожен, объекты Student останутся живыми.

2.2. Композиция

Композиция - это отношение "часть-целое", где объекты создаются внутри класса-контейнера и управляются им.

На содержащийся объект может ссылаться только содержащий его объект-контейнер, и первый должен быть удалён при удалении объекта-контейнера.

То есть объект-контейнер управляет временем жизни содержащихся объектов.

```
class CompositionEngine {
public:
        void start() {
                cout << "Engine started!" << endl;
        }
};

class CompositionCar {
private:
        CompositionEngine* engine;
public:
        CompositionCar() {
                engine = new CompositionEngine();
        }
        ~CompositionCar() {
                delete engine;
        }
        void start() {
                engine->start();
                cout << "Car started!" << endl;
        }
};
```

Объяснение: В этом примере CompositionCar содержит объект CompositionEngine, который создается внутри CompositionCar и удаляется вместе с ним.

Если CompositionCar будет уничтожен, CompositionEngine также будет уничтожен.

2.3. Обобщение

Обобщение - выражается наследованием, где дочерний класс является более конкретным (уточнённым/расширенным) по отношению к родительскому классу.

```
class Animal {
public:
        virtual void makeSound() {
                cout << "Animal sound" << endl;
        }
};

class Dog : public Animal {
public:
        void makeSound() override {
                cout << "Woof!" << endl;
```

```
        }
};
```

Объяснение: В этом примере Dog наследует от Animal и переопределяет метод makeSound.

Это показывает, что Dog является более конкретным типом Animal.

3. Реализация

Реализация - выражается через интерфейсы или абстрактные класса, где класс реализует методы интерфейса или абстрактного класса.

```
class ISoundMaker {
public:
        virtual void makeSound() = 0;
};

class Cow : public ISoundMaker {
public:
        void makeSound() override {
                cout << "Moo!" << endl;
        }
};
```

Объяснение: В этом примере Cow реализует интерфейс ISoundMaker, предоставляя свою версию метода makeSound.

Это показывает, что Cow реализует контракт, определенный в ISoundMaker