**Методические указания к выполнению практических заданий по курсу «Объектно-ориентированное программирование»**

Цикл состоит из 8 практических заданий, охватывающий разделы курса «Объектно-ориентированное программирование».

Для каждой работы приводится требования, предъявляемые к каждому проекту, который требуется реализовать. Формулируются цель работы и ограничения, в рамках которых эта цель должна быть реализована.

В качестве исходных данных для программирования в объектно-ориентированное парадигме (ООП) выступает объектная модель (ОМ) предметной области (ПрО), дополненная объектной библиотекой (ООБ) конструкторских классов (КК). ОМ в данном практическом цикле представлен в виде диаграммы классов (ДК), которую следует реализовать или разработать на основе описания задачи, заполнив классы ПрО необходимыми атрибутами (полями) и методами (операциями), обоснованно распределенными по областям видимости. В качестве КК на ДК используются стандартные классы библиотеки С++ и классы из ООБ Qt. На ДК используются следующие обозначения:



Диаграммы классов, представленные в заданиях задают **концепцию** построения приложений. На них могут быть не прорисованы некоторые конструкторские классы из ООБ. Здесь можно проявить творческий подход и использовать классы из соответствующих объектных библиотек. В отчетах по практическим работам следует представлять **полные диаграммы классов** со всеми используемыми классами иуказанием отношений между ними.

**Некоторые сведения о диаграммах классов**

Диаграмма классов занимает центральное место в логической модели ПО и используется для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Эта диаграмма может рассматриваться и строиться, как дальнейшее развитие концептуальной модели проектируемой системы.

Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы представляющие классы, дуги задают различные типы структурных отношений, которыми они связаны.

Класс (class) обозначает множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса. По мере проработки отдельных компонентов диаграммы, описания классов дополняются атрибутами и операциями.

Каждому атрибуту класса приписывается один из кванторов видимости. Квантор видимости может принимать одно из трех возможных значений и, соответственно, отображается при помощи специальных символов:

1. Символ "+" обозначает атрибут с областью видимости типа общедоступный *public).* Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма.
2. Символ "#" обозначает атрибут с областью видимости типа защищенный *(protected).* Атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса.
3. Символ "-" обозначает атрибут с областью видимости типа закрытый (*private).* Атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.

Вместо условных графических обозначений можно записывать соответствующее ключевое слово: *public, protected, private.*

Кроме внутреннего устройства или структуры классов на диаграмме указываются различные отношения между классами. Базовыми отношениями между классами или связями являются:

1. отношение зависимости (dependency relationship),
2. отношение ассоциации (assocfafion relationship),
3. отношение обобщения (inherlts from relationship),
4. отношение реализации (realization relationship).

Каждое из этих отношений имеет собственное графическое представление на диаграмме, которое отражает взаимосвязи между объектами соответствующих классов.

**Отношение зависимости** (dependency relationship).

Отношение зависимости является наиболее общей формой отношения. Оно имеет место всегда, когда изменение спецификации одного класса (класса-источника) может повлиять на работу другого класса (класса-клиента), но не наоборот. Отношение зависимости графически изображается пунктирной линией со стрелкой, направленной от класса-клиента зависимости к независимому классу или классу-источнику.

**Отношение ассоциации** (assoclation relationship).

Отношение ассоциации имеет место, когда объекты одного класса связаны с объектами другого класса таким образом, что можно перемещаться от объектов одного класса к другому. Отношение ассоциации обозначается сплошной линией, стрелка указывает на порядок следования классов.

**Отношение агрегации** (aggregation relationship).

Является частным случаем отношения ассоциации. Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности. Это отношение по своей сути описывает декомпозицию или разбиение сложной системы на более простые составные части, которые также могут быть подвергнуты декомпозиции, если в этом возникнет необходимость в последующем. Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой не закрашенный внутри ромб, который указывает на класс, представляющий собой "целое". Остальные классы являются его "частями".

**Отношение композиции** (composition relationship).

Является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого, и эти части существуют до тех пор, пока существует целое и прекращают свое существование вместе с целым. Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб, который указывает на класс, представляющий собой "целое". Остальные классы являются его "частями".

**Отношение обобщения** (inherits from relationship).

Является обычным таксономическим отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). Отношение описывает иерархическое строение классов и на следование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка. На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов. Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее отсутствие – на более специальный класс (класс-потомок или подкласс).

**Требования к реализации практических работ**

Для выполнения практических заданий нет ограничений для выбора языка, среды и объектной библиотеки. Однако проект должен быть реализован в точном соответствии с концепцией приведенных ДК (сохранение всех отношений указанных типов между классами). Допускаются расширение состава или замена используемых КК с сохранением их функционального назначения.

**Отклонение реализации от требующей реализации диаграммы классов влечет за собой снижение оценки по работе.**

Рекомендуемая среда разработки QtCreator и использование библиотеки классов Qt (язык программирования С++). На приведенных диаграммах классов в качестве конструкторских используются классы из этой библиотеки.

**Требования к оформлению отчета**

Отчет по практической работе выполняется 1 на бригаду и состоит из пояснительной записки в бумажном или электронном виде, к которому прилагается архив исходных файлов проекта.

Отчет должен содержать:

1. Титульный лист.

2. Задание на практическую работу.

3. Спецификации классов (понятий или предметов), которые требуется разработать:

- атрибуты (наименование, тип, область видимости, роль в описании понятия - для чего нужен?);

- методы (наименование, типы формальных параметров и возвращаемого значения, область видимости, для чего нужен?).

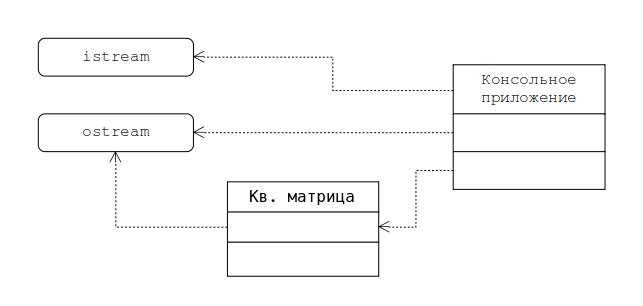
4. Диаграмму классов, дополненную атрибутами и методами.

5. Описание контрольного примера с исходными и ожидаемыми (расчетными) данными.

6. Скриншоты работы программы на контрольных примерах.

7. Выводы по выполнению работы.

**Практическая работа №1**

**Рис.1. Диаграмма классов работы №1**

Создать консольное приложение согласно представленной на рис.1 диаграмме классов, предназначенное для заданных вычислений над квадратной матрицей, заданной на **множестве вещественных чисел**. Для этого необходимо специфицировать пользовательские классы "Консольное приложение" и "Квадратная матрица", т.е. задать атрибуты и методы указанных классов, а также распределить их по существующим областям видимости. Спецификация классов и реализация их методов должна обеспечивать реализацию отношений, указанных на диаграмме классов. **В отчете представить аргументированное обоснование своего выбора**.

Приложение должно включать основной модуль (функция main), модуль «application» и модуль «matrix».

В **основном модуле** консольного приложения (*для языка С++ - это модуль с функцией main*) должен создаваться объект класса "Консольное приложение" и вызываться его метод, который предоставляет пользователю **меню команд** приложения.

Модуль **«application»** должен содержать спецификацию класса "Консольное приложение" и реализацию его методов. Один из его методов должен выводить в консоль меню команд приложения, включающее:

- команду, инициирующую ввод с консоли значений, задающих объект матрицы (до ввода в программе должна быть задана матрица по умолчанию);

- команду, инициирующую расчет определителя матрицы и вывод результатов расчета;

- команду, инициирующую формирования транспонированной матрицы и ее вывода в консоль;

- команду, инициирующую расчет ранга матрицы и вывод результатов расчета;

- команду, инициирующую представление в консоль текущего объекта матрицы;

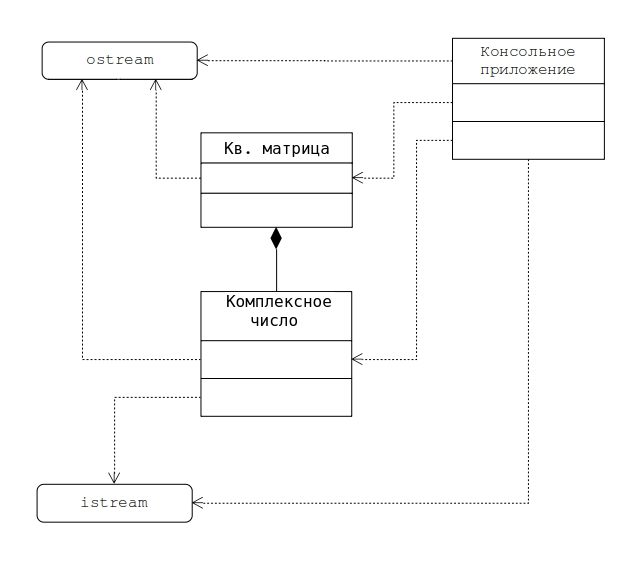
- команду выхода из приложения.

Модуль **«matrix»** должен содержать спецификацию класса "Квадратная матрица" и реализацию его методов, необходимых для достижения цели разрабатываемого приложения. Описание класса должно использовать вместо типа double (вещественное число, заданное в условии) абстрактный тип ***number***, описание которого должно задаваться в отдельном заголовочном файле number.h с помощью оператора **typedef double number** (для С++).

**ВАЖНО.** Основное требование к реализации класса **«matrix»** заключается в том, что она должна быть инвариантна (одна и та же для различных вариантов использования) ко множеству применимых объектов. Здесь задано множество определения элементов матрицы как вещественное, а может быть и множества целых, комплексных, рациональных и т.д. Для различных множеств при сохранении функциональности реализация класса должна быть одна и та же. Если для различных множеств менять реализацию, то теряется смысл ООП как парадигмы.

Требуется реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленным целям. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.

**Практическая работа №2**

**Рис.2. Диаграмма классов работы №2**

Создать консольное приложение, реализующее функции перечисленные в описании работы №1, но на множестве комплексных чисел.

Приложение должно включать основной модуль, модуль «application», модуль «matrix» и модуль «complex».

Для этого в проект лабораторной работы №1 следует добавить модуль с **собственным** (не брать из ООБ!) описанием и реализацией класса комрлексных чисел TComplex. Класс TComplex должен быть встроен в проект согласно диаграмме классов на рис.2. При этом основной модуль, модуль «application» и модуль «matrix» не должны изменяться. Изменения вносятся лишь в заголовочный файл number.h, где

**typedef double number;**

следует заменить на

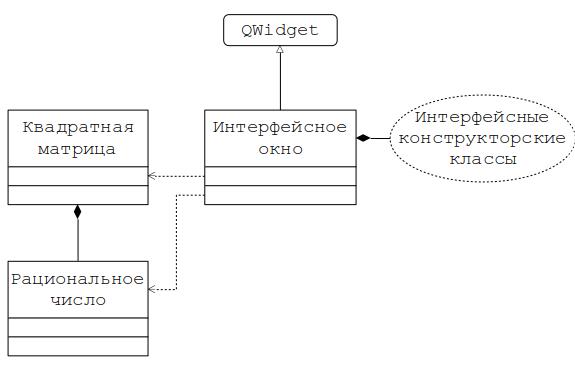
**#include «complex.h»**

**typedef TComplex number;**

В классе **TComplex** следует определить только те члены класса и спецификации, которые необходимы для совместимости модулей проекта и реализации отношений, приведенных в ДК объектной модели.

Реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленной цели. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.

**Практическая работа №3**

**Рис.3. Диаграмма классов работы №3**

Создать GUI приложение, реализующее функции перечисленные в описании работы №1, но на множестве рациональных чисел. Для этого требуется разработать и реализовать класс рациональных чисел.

Рациональное число — это **несократимая** дробь a/b, где a и b — целые, причем b>0.

Приложение должно включать основной модуль, модуль «interface», модуль «matrix», модуль «rational» и файл number.h:

**#include «rational.h»**

**typedef TRational number;**

.

Основной модуль main.cpp GUI приложения может иметь вид:

#include <QApplication>

#include "interface.h"

int **main**(int argc, char \*argv[])

{

QApplication a(argc, argv);

TInterface interface;

interface.show();

return a.exec();

}

При необходимости расширения функциональности класса «Квадратная матрица» следует только **дополнить** его протокол **без каких-либо изменений** уже существовавшей реализации.

Реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленным целям. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.

**Практическая работа №4**

|  |
| --- |
| Клиентская часть |
|  |
| Серверная часть |
|  |

**Рис.4. Диаграмма классов работы №4**

Создать распределенное приложение, включающее клиентскую и серверную части, взаимодействующие посредством сетевого обмена сообщениями.

Клиентская часть представляет собой GUI приложение, реализующее интерфейс аналогичный работе №3.

Серверная часть представляет собой консольное приложение, предназначенное для выполнения перечисленных в меню работы №1 функций над квадратной матрицей с рациональными элементами.

Диаграммы классов для клиентского и серверного приложений представлены на рис.4.

Реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленным целям. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.

**\*)** Класс "Коммуникатор" на основе сетевого протокола Udp с использованием конструкторских классов библиотеки классов Qt и пример его использования прилагаются к работе в виде исходных кодов.

**Практическая работа №5**

|  |
| --- |
| Клиентская часть |
|  |
| Серверная часть |
|  |

**Рис.5. Диаграмма классов работы №5**

Создать распределенное приложение, включающее клиентскую и серверную части, взаимодействующие посредством сетевого обмена сообщениями.

Клиентские и серверные части представляют собой приложения, реализованные в работе №4.

Отличие заключается в том, что класс «Квадратная матрица» делается наследников класса «Матрица» с произвольным числом строк и столбцов (отношение обобщения). Это влечет за собой перенос некоторых атрибутов и методов класса «Квадратная матрица» в родительский класс «Матрица», что и предстоит сделать в этой работе.

Реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленным целям. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.

**Практическая работа №6**

|  |
| --- |
| Клиентская часть |
|  |
| Серверная часть |
|  |

**Рис.6. Диаграмма классов работы №6**

Создать распределенное приложение, включающее клиентскую и серверную части, взаимодействующие посредством сетевого обмена сообщениями.

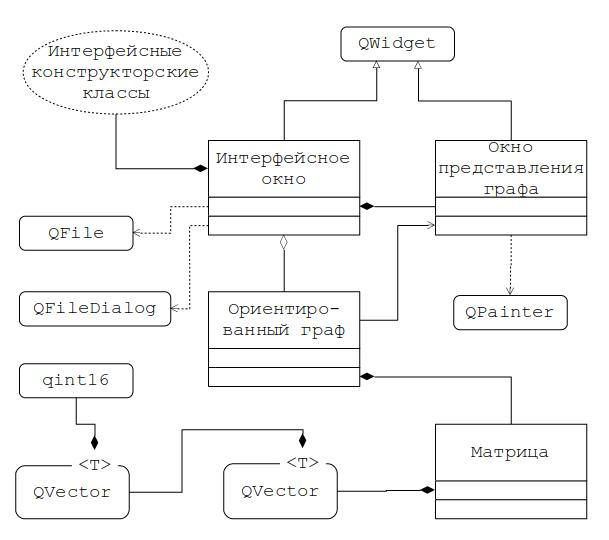
Клиентские и серверные части представляют собой приложения, реализованные в работе №5.

Клиентская часть модифицируется таким образом, что реализованные функции матриц могут исполняться по желанию пользователя на областях определения: вещественная, комплексная и рациональная.

Отличие серверной части заключается в том, что классы «Матрица» и «Квадратная матрица» параметризуются. Параметром класса делается абстрактный тип **number**, при этом файл number.h исключается из серверного приложения

Реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленным целям. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.

**Практическая работа №7**

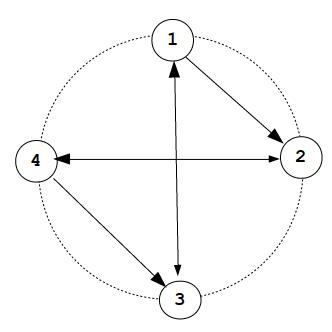


**Рис.7. Диаграмма классов работы №7**

Разработать GUI приложение, выполняющее функцию визуализации ориентированного графа, задаваемого матрицей смежности, представленной в виде файла, структуру которого требуется разработать. На рис.6 представлен макет диаграммы классов приложения, который требуется реализовать в приложении.

Основной функцией объекта класса "Интерфейсное окно" является выбор файла, который содержит данные об ориентированном графе. При чтении файла необходимо проверить корректность данных и в случае обнаружения ошибки необходимо сформировать соответствующее сообщение пользователю.

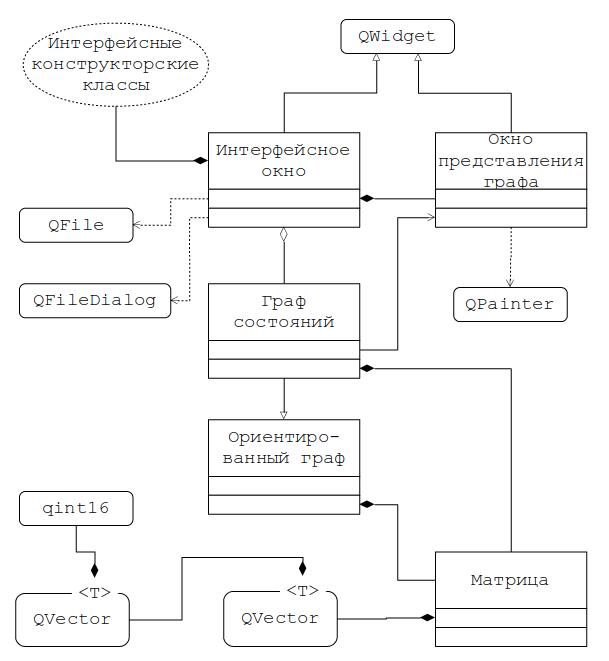
При корректности данных создается объект класса "Ориентированный граф", устанавливаются (если необходимо) связи между новым объектом и существующими, после чего граф отображается в соответствующем окне (объект класса "Окно представления графа"). Пример вида графа с 4 вершинами представлен ниже:



При выборе в интерфейсе другого графа (другого файла) старый должен заменяться на новый и перерисовываться.

Реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленным целям. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.

**Практическая работа №8**



**Рис.8. Диаграмма классов работы №8**

Разработать GUI приложение, выполняющее функцию визуализации графа состояний.

Граф состояний - это ориентированный граф, одна из вершин которого в каждый момент времени считается активной. Каждой дуге приписано некоторое событие, при возникновении которого происходит смена активной вершины.

Граф состояний описывается матрицей, число строк которой равно числу вершин, а число столбцов - числу событий. Элементом i-ой строки и j-го столбца является номер строки (т.е. соответствующая ей вершина графа), которая становится активной при возникновении j-го события, если при этом вершина i была активна.

На рис.8 представлен макет диаграммы классов приложения, который требуется реализовать в приложении.

Основной функцией объекта класса "Интерфейсное окно" является выбор файла, который содержит данные о графе состояний. При чтении файла необходимо проверить корректность данных и в случае обнаружения ошибки необходимо сформировать соответствующее сообщение пользователю.

Номер активной вершины также задается в интерфейсе.

При корректности данных создается объект класса "Граф состояний", устанавливаются (если необходимо) связи между новым объектом и существующими, после чего граф отображается в соответствующем окне (объект класса "Окно представления графа").

Активная вершина помечается цветом. При смене значения номера активной вершины должны происходить изменения в отображении.

В интерфейсе должна быть предусмотрена возможность инициирования любого из возможных событий. При их возникновении должен происходить переход в новую активную вершину, согласно графу, смена значения в интерфейсном окне и его перерисовка.

При выборе в интерфейсе другого графа (другого файла) старый должен заменяться на новый, номер активной вершины принимать исходное (корректное) значение и граф перерисовываться.

Реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленным целям. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.