МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МУРМАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
СВЕДЕНИЯ О СЕРТИФИКАТЕ ЭП
Сертификат: 36 22 ee ce 8d fd 1e 1d a3 ab
Владелец: Князева Мария Александровна

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

Б.О.09.02 Объектно-ориентированное программирование
(код и наименование дисциплины)

для направления подготовки 09.03.01 Информатика и вычислительная техника
(код и наименование направления подготовки /специальности)

направленности/профиля Программное обеспечение вычислительной техники и автоматизированных систем
(наименование направленности (профиля) образовательной программы)

Кафедра-разработчик: цифровых технологий, математики и экономики
(наименование кафедры-разработчика рабочей программы)

Мурманск
2021

Составитель - Лясникова Светлана Михайловна, доцент кафедры цифровых технологий, математики и экономики

Методические указания по освоению дисциплины рассмотрены и одобрены на заседании кафедры-разработчика

цифровых технологий, математики и экономики
название кафедры

21.06.2021г. протокол № 12
дата

ОГЛАВЛЕНИЕ

Общие организационно-методические указания

Обязательный минимум содержания дисциплины:

программирование в классах, роли класса, синтаксис класса, класс как тип данных, конструкторы, деструкторы, дружественные функции, структуры, отношения между классами, наследование, абстрактные классы, простое наследование, множественное наследование, виртуальные функции, виртуальные базовые классы, шаблоны функций и шаблоны классов, обработка исключительных ситуаций.

Целью дисциплины «Объектно-ориентированное программирование» является изучение основ классической теории объектно-ориентированного программирования: инкапсуляция, полиморфизм, абстракция, наследование, понятие класса, объекта, отношения между классами и др.

Задачей дисциплины «Объектно-ориентированное программирование» является изучение технологии объектно-ориентированного подхода в инструментальных языках и формирование практических навыков использования объектно-ориентированного программирования.

В результате изучения дисциплины студенты должны:

- знать и уметь использовать:

основы технологии объектно-ориентированного программирования; особенности построения объектно-ориентированных программных средств.

уметь реализовать принципы ООП при разработке программных средств; создавать диаграммы классов.

владеть приемами объектно-ориентированного решения различных задач.

Тематический план дисциплины

Таблица 1. Содержание учебной дисциплины

№ п\п	**Содержание разделов (модулей), тем дисциплины**	Количество часов, выделяемых на виды учебной подготовки	
		Лекции	ПР
1	2	3	4
II семестр			
Тема 1. Введение в ООП. Классы и ООП			
1	Для чего нужно ООП. Недостатки процедурного подхода.Объектно-ориентированный подход. Свойства ООП. Классы и ООП. Роли класса.Синтаксис класса. Поля класса. Методы класса. Методы свойства. Встроенное определение методов класса. Внешнее определение методов класса. Конструкторы. Создание и использование объектов. Деструкторы. Конструктор копирования. Указатель this. Константные поля. Статические элементы класса.	4	6
Тема 2. Перегрузка операций			
1	Ограничения на перегрузку. Прототип функции-операции. Перегрузка внешними функциями. Перегрузка операций методами класса.	2	0
Тема 3. Управляемый и неуправляемый код и данные			
1	Управляемый код и данные. Сборщик мусора. Ссылочные типы и типы значения.	0	0
Тема 4. Дружественные функции и классы			
1.	Дружественные функции: основные свойства, номенклатура. Ситуации, определяющие их необходимость и полезность. Наборы дружественных функций, дружественные классы. Объявление дружественной функции: синтаксис, размещение, семантика, требования к параметрам и типу возвращаемого значения. Размещение определения. Правила задания областей видимости. Вызов функции.	2	2
Тема 5. Отношения между классами. Простое наследование.			
1.	Отношения между классами. Отношения «является» и «имеет». Наследование: две роли. Простое наследование. Наследование членов базового класса в производном классе. Доступ к элементам базового класса в классе наследнике. Конструкторы, деструкторы и наследование. Порядок вызова конструкторов и деструкторов. Поля и методы при наследовании. Вложенные классы и наследование. Принцип подстановки. Закрытое наследование.	4	4
Тема 6. Виртуальные функции			
1	Зачем нужны виртуальные функции. Раннее (статическое) связывание. Определение виртуальных функций. Два типа полиморфизма в с++: статический полиморфизм, динамический полиморфизм. Правила описания и использования виртуальных функций.	4	2

	Переопределение и перегрузка виртуальных функций. Виртуальные функции в конструкторах и деструкторах. Чистые виртуальные функции. Виртуальные деструкторы. Чистые виртуальные деструкторы. Виртуализация внешних функций.		
Тема 7. Множественное наследование			
1	Множественное наследование. Принцип подстановки при открытом множественном наследовании. Принцип подстановки при открытом множественном наследовании. Виртуальное наследование. Принцип доминирования. Финальный класс. Размеры классов при множественном наследовании.	2	4
Тема 8. RTTI			
1	Составляющие механизма динамической идентификации типа. Типы преобразований. typeid().Класс type_info. Перекрестные ссылки. Мультиметоды. Двойной диспетчер для двух типов.	2	0
Тема 9. Шаблоны функций и классов			
1	Шаблоны функций: примеры. Параметры по умолчанию. Перегрузка шаблонов функций. Специализация шаблона функции. Определение шаблона класса. Объявление объекта класса. Внешнее определение методов. Параметры шаблона класса по умолчанию. Параметры шаблона – не типы. Специализация: частичная и полная. Ограничения. Поле-шаблон. Метод-шаблон. Параметр-шаблон. Шаблоны и наследование. Шаблоны и дружественность.	4	2
Тема 10. Диаграммы классов			
1	UML. Диаграмма классов. Изображение класса. Описание атрибута. Операции. Виды отношений между классами. Ассоциация. Агрегация. Композиция. Наследование. Зависимость.	2	2
Тема 11. Обработка исключительных ситуаций			
1	Принципы обработки исключений. Генерация исключений. Перехват и обработка исключений. Спецификация исключений. Исключения и конструкторы. Исключения и деструкторы. Стандартные исключения.	2	0
Тема 12. Стандартная библиотека шаблонов STL			
1	Библиотека STL. Шаблон vector. Возможности vector. Итераторы. Полезные методы. Последовательные контейнеры. Различные наборы операций для последовательных контейнеров. Инициализация контейнеров. Алгоритмы. Виды итераторов. Ассоциативные контейнеры.	2	4
Тема 13. Введение в паттерны проектирования			
	Основные элементы паттерна проектирования: имя, задача, решение, результаты. Классификация паттернов. Паттерн Singleton. Структура Singleton. Достоинства Singleton. Реализация Singleton.	1	4
Тема 14. Рефакторинг			
	Определение рефакторинга. Цель рефакторинга. Принципы рефакторинга.	1	4
Тема 15. Умные указатели			

	Умные указатели. Семантика перемещения. unique_ptr, weak_ptr, auto_ptr.	2	0
	Итого:	34	34

Перечень практических работ

Таблица 2. Перечень практических работ

№ п/п	Наименование и содержание практических работ	№ темы по таблице 1	Количество часов
1	2	3	4
1	«Решение СЛАУ Методом Гаусса»	1,2	4
2	«Объединения. Дружественные Функции»	4,5	2
3	«Нахождение площадей различных фигур»	7	2
4	«Множественное наследование. Разделение сиамских близнецов»	8	4
5	«Создание шаблона функции»	10	2
6	«Диаграммы классов»	11	2
7	«Контейнер vector»	13	4
8	«Потоки»	5,6,7	4
9	Семинар «Паттерн Decorator»	14	4
10	Семинар «Рефакторинг»	15	2
Итого			36

Методические указания к выполнению практических работ по темам

Практическая работа 1 , 2, 3

«Решение СЛАУ Методом Гаусса». «Разработка класса Стек».

Цель работы: *закрепление навыков создания классов, отработка понятия конструктор копирования.*

Краткие теоретические сведения, необходимые для выполнения работы

В теле класса могут быть объявлены:

- константы;
- поля;
- конструкторы и деструкторы;
- методы;
- события;
- классы (структуры, делегаты, интерфейсы, перечисления).

Поля класса – синтаксически являются обычными переменными языка и характеризуют свойства объектов класса:

- могут иметь любой тип, кроме типа этого же класса;
- могут быть описаны с модификатором const ;
- могут быть описаны с модификатором static;
- поля класса влияют на память.

Конструктор предназначен для инициализации объекта и вызывается автоматически при его создании. Две синтаксические особенности:

- Имя конструктора фиксировано и совпадает с именем класса.
- Для конструктора не задается возвращаемое значение

Свойства конструкторов

- Класс может иметь несколько конструкторов
- Конструктор, вызываемый без параметров, называется конструктором по умолчанию
- Если конструктор инициализирует все поля класса значениями, то это ПОЛНЫЙ конструктор
- Если программист не указал ни одного конструктора, компилятор создает его автоматически
- Конструкторы не наследуются
- Конструкторы нельзя описывать с модификаторами const , virtual, static

Деструктор служит для удаления объектов и освобождения ресурсов, занятых объектом, в первую очередь оперативной памяти.

Свойства деструктора

- Не имеет аргументов и возвращаемого значения
- Не может быть объявлен как const или static
- Не наследуется
- Может быть виртуальным
- Создается автоматически, если он не определен

Описывать в классе деструктор следует, когда объект содержит указатели на память, выделяемую динамически!

Конструктор копирования - специальный вид конструктора, получающий в качестве параметра указатель на объект этого же класса.

```
T(const T&A)
```

В С++ два типа копирования данных:

- поверхностное;

- глубинное.

Автоматически создаваемый конструктор копирования использует ПОВЕРХНОСТНОЕ КОПИРОВАНИЕ. Явно заданный конструктор копирования задает ГЛУБОКОЕ копирование.

Зачем нужен конструктор копирования

- При определении объекта для его инициализации используется другой объект;

```
T obj2 = obj1
```

- При передаче объекта в функцию в качестве параметра;
- Когда объект является значением, возвращаемым функцией obj = fun().

Задача 1

Создать класс для решения методом Гаусса системы линейных уравнений. В классе реализовать конструктор для глубокого копирования полей класса, которые являются указателями (поля для хранения матрицы коэффициентов, коэффициентов правых частей и вектора с результатом).

Реализовать перегрузку операции вывода. Операция бинарная. Должна возвращать ссылку. Важное: левым аргументом у операции вывода является объект cout типа ostream. Реализуя вывод методом класса не получится, чтобы cout был левым аргументом, т.к. левым аргументом любого метода является объект своего типа. А вызов операции будет выглядеть так: obj<<cout.

Отчет

Отчет должен содержать краткие теоретические сведения, использованные для разработки классов, разработанные классы с комментариями, результаты тестирования программ.

Практическая работа 4

« Дружественные Функции»

Цель работы: *приобрести навыки работы с дружественными функциями*

Краткие теоретические сведения, необходимые для выполнения работы

Свойства дружественных функций

- не связана с объектом и не является членом класса для которого она дружественна;
- задается и вызывается так же как и обычная глобальная функция;
- при вызове не получает указателя this;
- объявляется внутри класса к элементам которого ей нужен доступ;
- может быть обычной функцией или методом ранее определенного класса.

Задание 1

В ранее реализованном классе для решения СЛАУ методом Гаусса реализовать операцию вывода (<<) левым аргументом которой является системный объект, как дружественную функцию для использования операции вывода в «привычном» виде.

Отчет

Отчет должен содержать краткие теоретические сведения, использованные для работы, листинг программ с комментариями, результаты тестирования программ.

Максимальное количество баллов: 5

Практическая работа 5

«Нахождение площадей различных фигур»

Цель работы: *получить навык работы с виртуальными функциями, абстрактными классами*

Краткие теоретические сведения, необходимые для выполнения работы

Простое наследование

Классы A и B находятся в отношении «родитель – наследник», если при объявлении класса B класс A указан в качестве родительского класса, т.е. A родитель B, B наследник A. Производный класс имеет только одного родителя.

```
class имя_класса_потомка: [модификатор доступа] имя_базового_класса
{
   //тело класса
}
```

Порядок вызова конструкторов и деструкторов

- при создании объекта производного класса сначала вызывается конструктор базового, затем – производного;
- при отсутствии деструктора в производном классе система формирует деструктор по умолчанию;
- деструктор базового класса вызывается в деструкторе производного класса автоматически независимо от того, определен он явно или создан системой;
- деструкторы вызываются в порядке, обратном порядку вызова конструкторов.

Виртуальные функции – механизм с помощью которого можно узнать тип объекта во время выполнения программы.

Правила описания и использования виртуальных функций

- может быть только методом класса;
- любую перегружаемую операцию-метод класса можно сделать виртуальной;
- виртуальная функция наследуется;
- виртуальная функция может быть константной;
- если в базовом классе определена виртуальная функция, то метод производного класса с таким же именем и прототипом автоматически является виртуальным;
- конструкторы не могут быть виртуальными;
- статические методы не могут быть виртуальными;
- Деструкторы могут быть виртуальными.

Чистые виртуальные функции

```
virtual тип имя (параметры) = 0;
```

Класс, в котором есть хотя бы одна чистая виртуальная функция называется – **АБСТРАКТНЫМ.** Например:

```
class Animal
{
    public:
      virtual string kind() const =0;
};

class Elephant: public Animal
{
 public:
      virtual string kind() const
          {
          return "Elephant";
      }
};
```

Задача 1

Разработать абстрактный класс Shape, который определяет различные фигуры на плоскости. Этот класс должен содержать чисто виртуальный метод для вычисления площади фигур. Создать три производных класса Circle, Triangle, Rectangle с переопределенным методом для вычисления площади фигуры.

Отчет

Отчет должен содержать краткие теоретические сведения, использованные для работы, листинг программ с комментариями, результаты тестирования программ.

Практическая работа 6,7

«Множественное наследование. Разделение сиамских близнецов»

Цель работы: отработать понятие множественное наследование**,** изучить одну из проблем, возникающих при множественном наследовании и найти способ решения данной проблемы

Краткие теоретические сведения, необходимые для выполнения работы

Множественное наследование

```
class A{};
class B{};
class D: public A, public B
{};
```

Модификатор наследования может быть разный!

Задача 1

Создать класс A с методами a1, a2; классы B, C, E производные от A; класс D производный от B, C, E; класс F, производный от D. Показать порядок вызова конструкторов и деструкторов базовых классов при создании и уничтожении объектов класса F.

Задача 2 (Разделение сиамских близнецов)

Дано два класса различных разработчиков. Разработчик A представил базовый класс BaseA

```
class BaseA
{
    virtual int F (const char * );
};
```

Разработчик B представил базовый класс BaseB

```
class BaseB
{
    virtual int F (const char *);
};
```

Оба класса должны использоваться в качестве базовых. Кроме того функции F выполняют различные задачи.

Необходимо написать открытый производный класс MyClass от классов BaseA и BaseB, при этом нам требуется объект, который бы мог полиморфно использоваться функциями из библиотек обоих разработчиков.

Пример 1. Неверный

```
class MyClass: public BaseA, public BaseB
{
      int F(const char *p){//замещает метод F() из BaseA и из BaseB}
}
MyClass m;
BaseA *ca = new MyClass();
BaseB *cb = new MyClass();

ca->F("jhjhj");

cb->F("ljlll");
```

Вызываться будет всегда одна и та же версия F() из MyClass.

Решение заключается в изменении имен для функций.

```
class BaseA
{
   public:
    virtual int F (const char * p)
    {
```

```
    }
};

class BaseB
{
public:
    virtual int F (const char *p)
    {

    }
};

class BaseA2: public BaseA
{
   public:
          virtual int FA(const char *p) = 0;
   private:
          int F (const char *p) //замещаем наследованную функцию
          {
               return FA(p);
          }
};

class BaseB2:public BaseB
{
   public:
          virtual int FB(const char *p) = 0;
   private:
          int F (const char *p) //замещаем наследованную функцию
          {
               return FB(p);
          }
};
class MyClass: public BaseA2, public BaseB2
{
     public:
        int FA(const char *p)
        {
            //замещает метод F() из BaseA через BaseA2

        }

        int FB(const char *p)
        {
           //замещает метод F() из BaseB через BaseB2

        }

};

MyClass m;
BaseA *ca = new MyClass();
BaseB *cb = new MyClass();
```

```
ca->F("FA!!!FA");//вызов FA() из MyClass

cb->F("FB!!!FB");//вызов FB() из MyClass
```

Отчет

Отчет должен содержать краткие теоретические сведения, использованные для работы, листинг программ с комментариями, результаты тестирования программ.

Практическая работа 8

«Создание шаблона функции»

Цель работы: *получить навык создания шаблонов функций*.

Краткие теоретические сведения, необходимые для выполнения работы

Шаблоны функций

- сокращают исходный текст программы;
- уменьшают объем выполняемого файла (в работающую программу попадают только те варианты функций, которые реально были вызваны для выполнения).

Пример

```
template <class Any> //class = typename
void Swap(Any  &a, Any  &B)
{
   Any temp;
   temp = a;
   a=b;
   b=temp;
}
Any – параметр шаблона
```

Параметры по умолчанию задавать нельзя.

Задача 1

Создать шаблон функции для поиска максимального элемента в массиве произвольного типа.

Отчет

Отчет должен содержать краткие теоретические сведения, использованные для работы, листинг программы с комментариями, результаты тестирования программ.

Максимальное количество баллов: 1

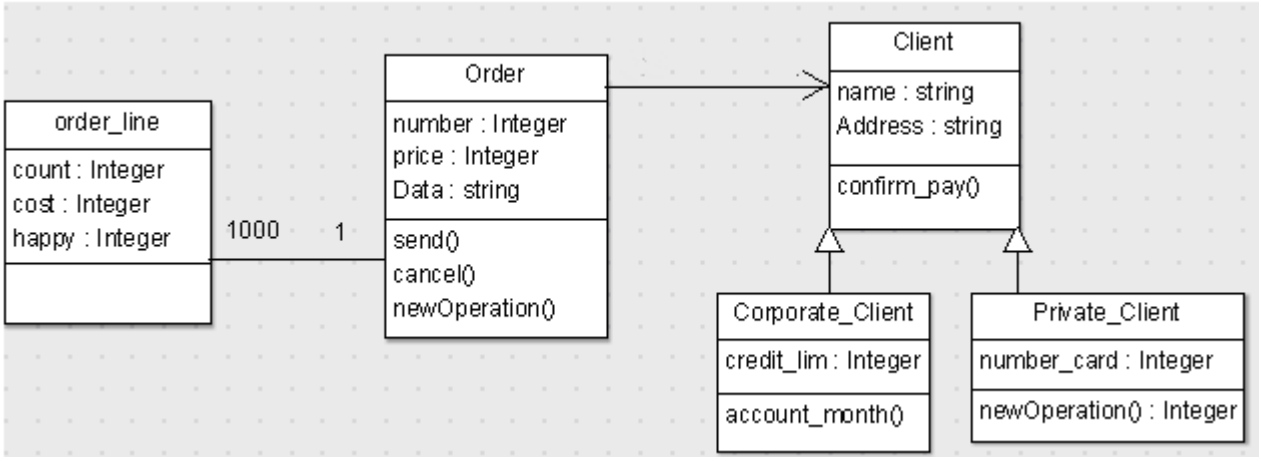
Практическая работа 9

«Диаграммы классов»

Цель работы: научиться читать диаграммы классов.

Задача

Ниже представлена диаграмма классов. Составьте код программы по представленной диаграмме классов.



Отчет

Отчет должен содержать краткие теоретические сведения, использованные для работы, листинг программы с комментариями.

Практическая работа 10, 11

«Контейнер vector»

Цель работы: научиться работать с контейнером vector из стандартной библиотеки шаблонов STL.

Краткие теоретические сведения, необходимые для выполнения работы

Шаблон vector – это набор однотипных значений, к которым можно обращаться в любом порядке. Используется как массив переменной длины.

Пример:

```
#include <vector>
vector <int> p(6);
for(int i=0; i<p.size(); i++)
{
    p[i] = i;       cout<<"p["<<i<<"]="<<p[i]<<endl;
}
```

Задача 1

В текстовом файле находится некоторое количество целых числе. Создайте программу, которая считывает эти числа в вектор и выводит эти числа на экран в порядке возрастания.

Отчет

Отчет должен содержать краткие теоретические сведения, использованные для работы, листинг программы с комментариями.

Практическая работа 12, 13

«Потоки»

Цель работы: научиться создавать отдельные потоки.

Краткие теоретические сведения, необходимые для выполнения работы

Qt предоставляет класс QThread для реализации потоков.

Поток — это независимая задача, которая выполняется внутри процесса и разделяет вместе с ним общее адресное пространство, код и глобальные данные.

Для создания потока в QT, необходимо создать производный класс от класса **QThread** и переопределить в нем чисто виртуальный метод *run()*, в котором должен быть реализован код, который будет исполняться в потоке.

Например:

```
class MyThread: public QThread
{
    public:
         void run()
         {
            //код, который будет исполняться в потоке
         }
}
```

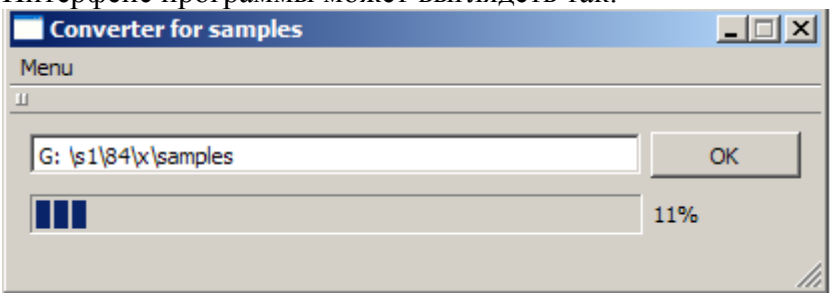
Далее необходимо вызвать метод *start()*, который вызовет реализованный метод `run()`.

```
MyThread thr;
thr.start();
```

Задача

В бинарном файле samples замените все отрицательные значения нулями (в файле находятся значения типа int). Создайте для этого отдельный поток, в котором проверяйте каждое значение и, если оно является отрицательным, замените его 0. Необходимо отобразить на ProgressBar сколько процентов осталось до окончания прохода по файлу.

Интерфейс программы может выглядеть так:



Класс для чтения файла и замены отрицательных значений нулями:

```
#ifndef READFILE_H
#define READFILE_H
#include <QThread>
#include <QFile>
#include <QProgressBar>
#include <QDebug>

class ReadFile : public QThread
{
    Q_OBJECT
    QString dir;

public:
    explicit ReadFile(QObject *pb = 0)
    {

    }
```

```
    void set_dir(QString directory)
    {
        dir = directory;

    }

    void run()
    {
          QFile file(dir);
          int coef = (file.size()/4)/100;
          if (file.open(QIODevice::ReadWrite))
          {
             QDataStream stream(&file);
             stream.setVersion(QDataStream::Qt_2_0);
             stream.setByteOrder(QDataStream::LittleEndian);
             int k = 0;

             while(!stream.atEnd())
             {

         //сюда необходимо вставить код для чтения текущего значения
         //и его замены на 0 если оно является отрицательными
                emit   progress_bar(1+(k/coef));//разберитесь     что
такое emit
                k++;

             }

             file.close();

          }

     }

signals:
    void progress_bar(int);

};
```

mainwindow.cpp

```
//….
void MainWindow::load_folder()
{
    QFileDialog fd;
    dir = fd.getOpenFileName();
    if  (dir=="")  ui->labelFileName->setText("Choose  file!");//если
пустая директория
    else
    {
       //директория
       dir.replace(QString('/'), QString("\\"));
       ui->labelFileName->setText(dir);

       rf = new ReadFile(ui->progressBar);
```

```
        rf->set_dir(dir);
        if (dir.contains("samples"))ui->ButtonOK->setEnabled(true);
        else ui->ButtonOK->setEnabled(false);
    }
}
void MainWindow::finished_thr()
{
    if (rf->isFinished())
    {
        ui->ButtonOK->setEnabled(false);
        delete rf;
    }
}
void MainWindow::buttonOK()
{
    QObject::connect(rf,SIGNAL(progress_bar(int)),ui->progressBar,
SLOT(setValue(int)));
    QObject::connect(rf,SIGNAL(finished()),this,
SLOT(finished_thr()));
    rf->start();
}
```

Практическая работа 14,15

Семинар «Паттерн Decorator»

Задание

Изучите следующие вопросы:
1. К какому классу паттернов относится Decorator.
2. Приведите структуру паттерна.
3. Покажите реализацию паттерна.
4. Приведите примеры программных решений с использованием данного паттерна.

Практическая работа 16, 17

«Рефакторинг: самотестирующийся код»

Краткие теоретические сведения

Для любого класса можно написать тест. Например, Qt предоставляет для этого модуль QtTestLib.

Называть тестирующий класс лучше следующим образом: префикс Test, а затем имя тестируемого класса. Например, тестирующий класс для класса Square, назовем Test_Square. Также тестовые методы и слоты лучше называть именами тестируемых методов.

Тестовый класс должен быть унаследован от класса **QObject** и содержать в своем определении макрос **QObject**, что позволит вызывать слоты класса при исполнении, включая его тестовые слоты в секции **private.**

Макрос QCOMPARE(), который можно применять в методах тестирующего класса, принимает два аргумента – это полученный и ожидаемый результат. Затем макрос

сравнивает эти результаты. Если значения не совпадают, то тогда исполнение тестового метода **прерывается** сообщением о не пройденном тесте.

Qt предоставляет макрос QTEST_MAIN() для исполнения каждого теста, этот макрос заменяет main.

В **pro-файле** должно содержаться следующее:

```
CONFIG+=qtestlib

win32:TARGET = ../TestLib
```

Рассмотрим пример. Напишем простой класс `MyClass`, содержащий два метода, которые мы подвергнем тестированию.

```
#ifndef MYCLASS_H
#define MYCLASS_H
class MyClass
{
  public:
    int Parabola(int x)
    {
        return x*x;
    }
    int LineFunc(int k,int x,int b)
    {
        return k*x+b;
    }
};

#endif // MYCLASS_H
```

Класс для проверки методов класса MyClass.

```
#ifndef TEST_MYCLASS_H
#define TEST_MYCLASS_H
#include <QtTest>
#include "myclass.h"
#include <QObject>
class Test_MyClass : public QObject
{
    Q_OBJECT
private slots://обязательно должны быть закрытыми!!!
    void Parabola()
    {
        MyClass m;
        QCOMPARE(m.Parabola(3),9);
        QCOMPARE(m.Parabola(1),1);
        QCOMPARE(m.Parabola(0),0);
        QCOMPARE(m.Parabola(2),4);
        QCOMPARE(m.Parabola(8),64);
    }
    void LineFunc()
    {
        MyClass m;
        QCOMPARE(m.LineFunc(0,2,5),5);
        QCOMPARE(m.LineFunc(2,5,5),15);
        QCOMPARE(m.LineFunc(100,3,5),305);
        QCOMPARE(m.LineFunc(2,3,4),10);
        QCOMPARE(m.LineFunc(7,1,3),10);
    }
   };
#endif // TEST_MYCLASS_H
```

main.cpp

```
#include <QtCore/QCoreApplication>
#include "test_myclass.h"
```

```
QTEST_MAIN(Test_MyClass)

/*int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    return a.exec();
}*/
```

После запуска программы, если все тесты прошли успешно, Вы должны увидеть следующее:

```
********* Start testing of Test_MyClass *********
Config: Using QTest library 4.7.1, Qt 4.7.1
PASS : Test_MyClass::initTestCase()
PASS : Test_MyClass::Parabola()
PASS : Test_MyClass::LineFunc()
PASS : Test_MyClass::cleanupTestCase()
Totals: 4 passed, 0 failed, 0 skipped
********* Finished testing of Test_MyClass *********
```

Теперь сделайте, какой-нибудь тест некорректным. Например:

```
 QCOMPARE(m.LineFunc(7,1,3),0);
```

Тогда, Вы увидите следующее:

```
********* Start testing of Test_MyClass *********
Config: Using QTest library 4.7.1, Qt 4.7.1
PASS : Test_MyClass::initTestCase()
PASS : Test_MyClass::Parabola()
FAIL! : Test_MyClass::LineFunc() Compared values are not the same
Actual (m.LineFunc(7,1,3)): 10
Expected (0): 0
debug\/../../untitled8/test_myclass.h(33) : failure location
PASS : Test_MyClass::cleanupTestCase()
Totals: 3 passed, 1 failed, 0 skipped
********* Finished testing of Test_MyClass *********
```

Тестовые данные можно поместить в отдельный тестирующий слот. Например:

```
#ifndef TEST_MYCLASS_H
#define TEST_MYCLASS_H
#include <QtTest>
#include "myclass.h"
#include <QObject>
class Test_MyClass : public QObject
{
    Q_OBJECT
private slots:
    void LineFunc_data()
    {
        QTest::addColumn<int>("k");
        QTest::addColumn<int>("x");
        QTest::addColumn<int>("b");
        QTest::addColumn<int>("res");
        QTest::newRow("LineFunc_test1") << 0 << 2 << 5 <<5;
        QTest::newRow("LineFunc_test2") << 2 << 5 << 5 << 15;
        QTest::newRow("LineFunc_test3") << 100 << 3 << 5 << 5;
        QTest::newRow("LineFunc_test4") << 2 << 3 << 4 << 10;
        QTest::newRow("LineFunc_test5") << 7 << 1 << 3 << 0;
    }
    void Parabola()
    {
        MyClass m;
        QCOMPARE(m.Parabola(3),9);
        QCOMPARE(m.Parabola(1),1);
        QCOMPARE(m.Parabola(0),0);
        QCOMPARE(m.Parabola(2),4);
        QCOMPARE(m.Parabola(8),64);
    }
    void LineFunc()
```

```
    {
        MyClass m;
        QFETCH(int, k);
        QFETCH(int, x);
        QFETCH(int, b);
        QFETCH(int, res);
        QCOMPARE(m.LineFunc(k, x, b), res);
    }
   // #include "test.moc"
};
#endif // TEST_MYCLASS_H
```

В слоте `LineFunc_data()` создается таблица тестовых данных, для этого используются методы `QTest:addColumn()` и `QTest::newRow()`. Макрос `QFETCH()` необходим для создания локальных переменных *k, x, b*, при этом имя переменной соответствует переменной в тестовых данных.

Результат выполнения программы:

```
Запускается D:\Qt\qtcreator-2.2.0\untitled8-build-desktop\TestLib.exe...
********* Start testing of Test_MyClass *********
Config: Using QTest library 4.7.1, Qt 4.7.1
PASS : Test_MyClass::initTestCase()
PASS : Test_MyClass::Parabola()
FAIL! : Test_MyClass::LineFunc(LineFunc_test3) Compared values are not the same
Actual (m.LineFunc(k, x, b)): 305
Expected (res): 5
debug\/../../untitled8/test_myclass.h(48) : failure location
FAIL! : Test_MyClass::LineFunc(LineFunc_test5) Compared values are not the same
Actual (m.LineFunc(k, x, b)): 10
Expected (res): 0
debug\/../../untitled8/test_myclass.h(48) : failure location
PASS : Test_MyClass::cleanupTestCase()
Totals: 3 passed, 2 failed, 0 skipped
********* Finished testing of Test_MyClass *********

D:\Qt\qtcreator-2.2.0\untitled8-build-desktop\TestLib.exe завершился с кодом 2
```

Задание

Проверьте метод для вычисления значения многочлена для заданного аргумента в созданном Вами ранее классе «Многочлен».

Источники: http://qt-doc.ru.

Список цитируемой литературы

При разработке материалов лабораторных занятий **использовались и цитировались** следующие источники:

1. Саттер, Герб Решение сложных задач на C++. Серия C++ In-Depth, т.4: Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 400 с.
2. Лаптев, В.В. C++.Объектно-ориентированное программирование: Учебное пособие. – Спб: Питер, 2008. – 464 с.
3. Леен Аммераль STL для программистов на C++. – М: ДМК, 1999. – 240 с.
4. Прата Стивен Язык программирования C++. Лекции и упражнения, 5-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2007. – 1184 с.
5. Вендоров А.М. Проектирование программного обеспечения экономических информационных систем. – М.: Финансы и статистика, 2002. – 349 с.

6. Павловская Т.А., Щупак Ю.А. С++. Объектно-ориентированное программирование: Практикум. – Спб: Питер, 2006. – 265 с.
7. Язык С++: Учеб. пособие/ И. Ф. Астахова, С.В. Власов, В.В Фертиков, А.В. Ларин. – Мн.: Новое знание, 2003. – 203 с.
8. Гамма Э., Джонсон Р., Хелм Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – М.: Питер, 2010. – 368 с.
9. http://qt-doc.ru/qthread-potoki-v-qt.html.
10. Стрикелева, Л. В. Программирование. Курс лекций (РФиКТ, БГУ (Минск), 1-ый курс), 2012, http://elib.bsu.by/handle/123456789/8654.

Рекомендуемая литература

- Антони Синтес Освой самостоятельно объектно-ориентированное программирование за 21 день. - М.: Вильямс, 2002.
- Гамма Э., Джонсон Р., Хелм Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – М.: Питер, 2010. – 368 с.
- Грэхем Иан Объектно-ориентированные методы. Принципы и практика = Object-Oriented Methods: Principles & Practice. - М.: Вильямс, 2004.
- Джосьютес Н. С++. Стандартная библиотека. Для профессионалов. – Спб: Питер, 2004. – 730 с.
- Иванова Г.С, Ничушкина Т.Н., Пугачев Е.К. Объектно-ориентированное программирование. -М.: Изд-во МГТУ им. Н.Э. Баумана, 2001.
- Лафоре Р. Объектно-ориентированное программирование в С++. – Питер, 2004.
- Павловская Т. А. С/С++. Программирование на языке высокого уровня. - ЗАО Издательский дом «Питер», 2003.
- Прата Стивен Язык программирования С++. Лекции и упражнения, 5-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2007. – 1184 с.
- Скотт Мейерс Эффективное использование STL. – Питер, 2002.
- Стрикелева, Л. В. Программирование. Курс **лекций** (РФиКТ, БГУ (Минск), 1-ый курс), 2012, http://elib.bsu.by/handle/123456789/8654.
- Эрик Гамма, Ральф Джонсон, Ричард Хелм, Джон Влиссидес Приемы объектно-ориентированного проектирования. Паттерны проектирования. - Питер, 2007.
- Язык С++: Учеб. пособие/ И. Ф. Астахова, С. В. Власов, В.В. Фертиков, А. В. Ларин. – Мн: Новое знание, 2003. – 203 с.
- http://qt-doc.ru/qthread-potoki-v-qt.html