Лекция 1

1. Введение

1.1. Цель дисциплины «Программирование на основе классов и шаблонов» (ПКШ) – освоение методов ООП в области разработки информационных систем.  
В процессе обучения будет разрабатываться библиотека классов для макета СУБД (Система Управления Базами Данных).

1.2. Объем дисциплины ПКШ – 180 часов, в том числе:

* лекции – 34 часа.
* семинары – 17 часов.
* лабораторные работы (ЛР) – 34 часа
* самостоятельная работа студента (СРС) – 95 часов.k

Дисциплина завершается экзаменом.

1.3. Структура курса.  
Курс ПКШ разделен на два модуля.  
Модуль 1. Разработка библиотеки классов для макета СУБД.

Модуль 2. Использование библиотеки классов при разработке информационных систем.

Параллельно с дисциплиной ПКШ в соответствии с программой обучения проводится Лабораторно-вычислительный практикум (ЛВП). ЛВП – это самостоятельная дисциплина, которая заканчивается зачетом с оценкой. Объем дисциплины ЛВП – 74 часа (ЛР - 34 часа, СРС - 40 часов).

Цель ЛВП – освоение процесса создания программного обеспечения (ПО) на примере разработки программы и эксплуатационной документации для информационной системы (ИС). В разрабатываемой ИС должна использоваться создаваемая вами в курсе ПКШ библиотека классов.

1.4. Содержание дисциплины ПКШ.

На примере разработки макета СУБД рассматривается реализация в ООП основных принципов программирования:

* Разделение программы на модули, которые могут быть отлажены независимо друг от друга и затем многократно использованы. Реализуется с помощью классов.
* Инкапсуляция (сокрытие сложности реализации и простота использования). Реализуется с помощью управления доступом к членам класса.
* Повторное использование кода:   
  - наследование классов;   
  - включение в класс объектов других классов (композиция);   
  - использование шаблонов классов;   
  - STL (Стандартная Библиотека Шаблонов).
* Обобщенное программирование:   
  - виртуальные функции;   
  - полиморфизм;

- шаблоны классов.

* Управление памятью.
* Обработка ошибок (исключения).

1.5. Порядок изложения материала дисциплины ПКШ.  
Порядок изложения материала курса будет, по возможности, придерживаться принятому порядку разработки программного обеспечения (ПО). Это позволит совместить в одном семестре изучение дисциплины ПКШ с выполнением заданий ЛВП .

*При изучении дисциплины мы будем использовать метод проектов. Метод проектов – это одна из форм интерактивную обучения. Суть интерактивного обучения хорошо представляется китайской притчей:   
«Скажи мне – и я забуду; покажи мне – и я запомню; дай сделать – и я пойму».*

***Разработка проекта (метод проектов)*** *— это способ достижения дидактической (от общего к частному, в отличии от индукции – от частного к общему) цели через детальную разработку проблемы (технологию), которая должна завершиться вполне реальным, осязаемым практическим результатом, оформленным тем или иным образом.  
 Основное предназначение метода проектов состоит в предоставлении обучающимися возможности самостоятельного приобретения знаний в процессе решения практических задач или проблем, требующего интеграции знаний из различных предметных областей.*

* *В основе метода проектов лежит развитие познавательных навыков обучающихся, умений самостоятельно конструировать свои знания, умений ориентироваться в информационном пространстве.*
* *Метод проектов всегда ориентирован на самостоятельную деятельность обучающихся - индивидуальную, групповую, которую обучающиеся выполняют в течение определенного отрезка времени.*
* *Метод проектов всегда предполагает решение какой-то проблемы.*
* *Результаты выполненных проектов должны быть, что называется, "осязаемыми", - конкретным результатом, готовым к использованию.*

Материал курса будет излагаться по мере необходимости его использования в процессе разработки библиотеки классов для макета СУБД.

Такое совмещение, с одной стороны, поможет лучше усвоить материал курса, но с другой стороны, потребует предварительного “поверхностного” объяснения реализации на С++ некоторых элементов ООП. Глубина и объем изложения должны обеспечить возможность использования их в данный момент в разработке макета СУБД. Иными словами, не будем тратить время на глубокое и широкое изучение элементов языка, иначе не останется времени на разработку программы, то есть на проверку правильности усвоения излагаемого материала на практике. Более глубокое изучение элементов языка будет выполняться во второй половине семестра.

# 2. Понятие «класс»

В настоящее время существуют два основных подхода к разработке программ:   
- ООП – программирование, ориентированное на манипуляцию объектами предметной области;  
- процедурное программирование (ПП), ориентированное на разработку алгоритма, манипулирующего функциями. ПП вы изучали в курсе «Основы программирования».

//----------------------------------------------------------------------

### Другие концепции программирования.

Кроме рассмотренных выше парадигм (концепций) программирования широкое распространение получили следующие парадигмы.

Декларативное программирование - это [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в которой задаётся [спецификация](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) решения задачи, то есть описывается, что представляет собой проблема и ожидаемый результат.

Пример: язык SQL, в котором просто создаем запрос, в котором диктуем, какие данные нам нужны и откуда их взять. А план выполнения запроса сервер построит сам.

### Понятие объектно-ориентированного проектирования.

Существует два понятия: объектно-ориентированное проектирование и объектно-ориентированное программирование.

Оба понятия неразрывно связаны друг с другом. Чтобы не путать эти понятия, давайте разберемся, что первично.

При создании программ с использованием объектного подхода выполняется:

1. OOA (object oriented analysis) - объектно-ориентированный анализ,
2. OOD (object oriented design) - объектно-ориентированное проектирование,
3. OOP (object oriented programming) - объектно-ориентированное программирование.

### Соотношение понятий проектирования и программирования.

Исходя из алгоритма создания программ на основе объектного подхода, после выполнения каждого шага мы получаем следующие результаты.

1. Объектно-ориентированный анализ позволяет получить модель предметной области.
2. Объектно-ориентированное проектирование - разработать систему классов.
3. Объектно-ориентированное программирование – реализовать систему классов и написать программу на языке программирования.

//---------------------------------------------------------------------

Использование при разработке сложных информационных систем ООП вместо ПП позволяет в большинстве случаев реализовать один из основных принципов проектирования вообще (не только разработки программ) - принцип KISS («keep it simple, stupid» или «keep it small and simple»). Принцип KISS утверждает, что большинство систем работают лучше всего, если они остаются простыми, а не усложняются. Поэтому в области проектирования простота должна быть одной из ключевых целей, и следует избегать ненужной сложности (small and simple).

ООП – это программирование, ориентированное на использование классов. Объектно-ориентированный язык программирования С++, который является развитием языка С, называют *«С с классами»*. Использование классов в языках программирования позволяет более естественно реализовывать принцип KISS при разработке больших программных систем.

Что же такое «класс»?   
Класс – это тип данных, определяемых пользователем. Исторически «класс» в *С++* является расширением типа данных «структура» в *С*. Класс представляет в программе описание абстракции некоторого объекта в предметной области задачи. В абстрактном объекте учитываются только те характеристики, которые нужны для решения задачи.   
Элементами класса являются поля, хранящие значения свойств объекта, и методы (функции - члены класса). Методы реализуют действия, которые должен выполнять объект, или действия, которые изменяют состояние объекта.  
 *Состояние объекта определяется совокупностью значений его свойств.*

Классы позволяют легко разделить программу на небольшие модули, отладить их и скрыть сложность реализации внутри класса, оставив открытым достаточно простой интерфейс.

Доступ к элементам класса в программе определяется спецификаторами доступа ***private, public, protected.***Структура в С++ - это частный случай класса, в котором все элементы доступны, то есть по умолчанию имеют спецификатор доступа ***public***, а в классе значение спецификатора доступа по умолчанию – ***private***.   
Внешним функциям доступны только те элементы, которые имеют спецификатор доступа ***public***, а функции-члены класса имеют доступ ко всем элементам класса.   
Спецификатор ***protected*** используется при наследовании и будет рассмотрен позже.  
Общедоступные элементы класса определяют его интерфейс, а закрытые – внутреннюю структуру (реализацию).

Пример описания класса DBTableTxt приведен в Приложении 1.

Как и при использовании структуры, следует различать *описание класса* и *определение объектов класса*. Сначала нужно описать (определить) *класс*, а затем можно создавать *объекты* (экземпляры) класса.

При создании объектов класса память выделяется только под поля класса. Методы класса, как и функции в С++, хранятся в памяти в одном экземпляре.

Когда мы создаем экземпляры (переменные) класса, мы не можем присвоить им значения в самом определении класса. Компилятор выдаст ошибку, также как это было и при использовании структур. Поэтому в классе необходимо иметь отдельный метод (так как поля класса обычно имеют спецификатор доступа private), с помощью которого будет происходить инициализация элементов. Для этого в определение класса вводится конструктор (от слова construct — создавать). Конструктор – это специальный метод класса, вызов которого компилятор вставляет в код программы всякий раз, когда создается экземпляр класса. Для того, чтобы можно было создавать массивы, в определении класса всегда должен присутствовать конструктор по умолчанию, для вызова которого не нужно указывать параметры. Этот конструктор должен вызываться для каждого элемента массива и в С++ в этом случае нет возможности передавать ему параметры, можно только использовать значения, установленные по умолчанию.

В отличии от конструктора, деструктор (от слова destruct — разрушать) — специальный метод класса, который служит для освобождения выделенной объекту с помощью операции *new* динамической памяти. Деструктор всегда вызывается по умолчанию при выходе объекта из области существования. Так как он не принимает никаких параметров, то в классе может существует только один деструктор.

Важно запомнить:   
- конструктор и деструктор, мы всегда объявляем в разделе public;   
- при объявлении конструктора, тип данных возвращаемого значения не указывается, в том числе - void;   
- у деструктора также нет типа данных для возвращаемого значения; к тому же деструктору нельзя передавать никаких параметров;   
-имена конструкторов и деструктора должно совпадать с именем класса;   
- перед именем деструктора ставится приставка ‘~’ ;

В классе допустимо создавать несколько конструкторов, если это необходимо. Компилятор будет их различать по передаваемым параметрам (как при перегрузке функций). Если мы не передаем в конструктор параметры, он считается конструктором по умолчанию.

В классе может быть объявлен только один деструктор, так как его имя фиксировано и он не может иметь параметров. Компилятор всегда создает конструктор и деструктор по умолчанию и вставляет в программу их вызов, но они не выполняют никаких действий. Чтобы они что-то делали, их нужно перегрузить.

Для вызова метода используется операция ‘.’, если метод вызывается значением объекта, или ‘->’, если метод вызывается указателем на объект.   
Вспомним, что в программе может быть объявлено несколько объектов одного класса, а методы класса существуют в одном экземпляре. Для идентификации объекта, то есть определения, какой объект вызвал метод класса, всем методам класса в качестве первого параметра передается адрес вызывающего объекта – указатель на этот объект ***this***. Так как этот параметр передается всегда при вызове метода, то он явно не указывается (передается по умолчанию), но его можно использовать в методах класса.

Кроме функций-членов класса, в описании класса могут присутствовать функции-друзья класса. Они имеют спецификатор *friend* перед прототипом функции и могут располагаться в любой области описания класса. Им не передается указатель ***this,*** но они имеют доступ к закрытым элементам класса. Адрес объекта *должен* передаваться им в первом параметре.   
Функции-друзья класса определяются вне класса, как глобальные функции, и используются для перегрузки методов класса. Мы будем их использовать в ЛР при разработке собственных методов классов путем перегрузки методов статической библиотеки классов для макета СУБД (*dbmsLib*), которая имеется на кафедре.

# 3. Пример разработки описания класса.

Предположим, что вы создали небольшую фирму по разработке информационных систем (ИС) для ленивых студентов, которым нужно получить зачет по ЛВП.

***Краткая информация по ИС.***  
*В информационных системах данные хранятся на дисках, чтобы их можно было бы многократно использовать в течение длительного промежутка времени. Обычно данные хранятся в виде связанных между собой таблиц. Такие данные называются реляционными базами данных (БД). Для связи между таблицами используются так называемые ключевые поля (ключи). Значение ключа позволяет однозначно идентифицировать экземпляр объекта, например, номер зачетной книжки однозначно идентифицирует студента. В ВУЗе не может быть двух студентов с одинаковыми номерами зачетных книжек. Другим примером ключа может служить номер паспорта гражданина.  
Каждая таблица хранит данные* ***одного*** *из объектов предметной области, для которой разрабатывается информационная система.*

Предположим, что к вам поступил заказ на разработку двух ИС: «Библиотека» и «Магазин» и вы ожидаете поступление других заказов на аналогичные системы.

Разработку ИС начинают с выделения объектов предметной области и словесного описания их взаимодействия в процессе выполнения указанных в задании функций. Например, ИС «Библиотека» должна помочь библиотекарю выполнять следующие функций:   
- выдача книг студентам;   
- проверка наличие свободных экземпляров книги;   
- определение студентов, не вернувших вовремя книгу.  
В ИС «Магазин» должны выполняться следующие функции:   
- определение наличия товара на складе;   
- определение дохода за определенный период времени;   
- выдача заказа.

Пусть в результате словесного описания выполнения указанных в задании функций мы определили объекты, их параметры и методы, позволяющие выполнить эти функции, и разработали таблицы для хранения состояний объектов в БД «Библиотека» и «Магазин».   
В таблицах БД можно ограничиться 4-мя типами данных:

string, int, double и DBDate.

- int и double – *встроенные* в С++ типы;

- DBDate – тип для представления времени в БД. Мы создадим его сами для выполнения операций со временем при работе с БД;

- string – тип данных из *стандартной библиотеки* С++.

***База данных библиотеки*** ***Library*** содержит 3 таблицы с данными:

Abonements, Students и Books.

Имена столбцов и типы данных в столбцах таблиц БД Library:

*Таблица Students.* Имена столбцов: StudentID, Name, Group.

Типы данных в столбцах таблицы:

StudentID - int (имя типа Int32, длина 8);

Name – string (имя типа String, длина 16);

Group– string (имя типа String, длина 8);

*Таблица Books*. Имена столбцов: Author, BookID, Title, Quantity, Number, где:

Quantity – количество поступивших в библиотеку экземпляров данной книги.

Number – количество выданных экземпляров данной книги.

Типы данных в столбцах таблицы:

Author - string (имя типа String, длина 12);   
BookID - int (имя типа Int32, длина 8);

Title – string (имя типа String, длина 16);   
Quantity - int (имя типа Int32, длина 8);

Number - int (имя типа Int32, длина 8);

*Таблица Abonements*. Имена столбцов: StudentID, BookID, OutDate, InDate, где:

DateOut, DateIn – дата выдачи и дата возврата книги.

Типы данных в столбцах таблицы:

StudentID - int (имя типа DBDate, длина 12);

BookID - int (имя типа Int32, длина 8);

OutDate - DBDate (имя типа DBDate, длина 12);

InDate - DBDate (имя типа DBDate, длина 12);

***Краткая информация по СУБД.***  
*Просматривая описания действий библиотекаря или продавца магазина, можно заметить, что эти действия реализуются в разных ИС с помощью небольшого набора методов, вызываемых различными объектами: чтение параметров объекта из файла в ОП, изменение значений полей экземпляров объекта, добавление и удаление экземпляров объектов, сохранение состояния объектов в файле, поиск и выборка экземпляров объектов по запросу (по значению полей данных). Более того, различные объекты ИС в СУБД можно представить одним «обобщенным» объектом –таблицей БД (DBTable). Чтобы использоваться для хранения состояний различных объектов, эта таблица должна иметь заголовок с переменным числом столбцов и должна позволять хранить в столбцах данные различных типов. Если эти условия выполнить, то все задания на разработку различных информационных систем можно будет решать в одной программной системе.  
Такие системы давно существуют. Это СУБД – Системы Управления Базами Данных.  
Программное обеспечение СУБД устанавливается на специализированные ЭВМ – серверы БД. СУБД может работать с БД огромных размеров и одновременно обслуживать большое число удаленных пользователей.*

В курсе ПКШ мы будем разрабатывать свой, упрощенный вариант СУБД - макет СУБД. Упрощения будут состоять в том, что макет СУБД будет установлен на ПЭВМ, может одновременно работать только с одним пользователем и использует таблицы небольшого размера, которые могут целиком сохраняться в ОП в объектах типа DBTable.   
В начале предположим, что таблицы хранятся в текстовых файлах формата CSV, а затем мы подключим к макету СУБД базу данных с таблицами, хранящимися в бинарных файлах.

Класс для представления текстовых таблиц в макете СУБД назовем DBTableTxt.  
Состав необходимых полей и функций-членов класса должен обеспечивать выполнение индивидуальных заданий ЛВП.

Приведем описание этого класса (оно используется в ЛР).

Приложение 1.

Файл DBTableTxt.h

//============ Класс "Таблица макета СУБД"==============

//--------DBType-перечисление типов полей таблиц БД------

enum DBType {

NoType,

Int32,

Double,

String,

Date

};

enum Condition{Undefined,Equal,NotEqual,Less,Greater,LessOrEqual,GreaterOrEqual};

const int LENGTH = 24;//длина имени таблицы и имени столбца.

//ColumnDesc - описание данных в столбце таблицы

//данные типа string имеют одинаковую длину (length) внутри //одного столбца,

//но в разных столбцах их максимальный размер может отличаться

struct ColumnDesc {

char colName[LENGTH];//имя столбца

DBType colType;//тип данных в столбце таблицы

int length; //максимальное число символов, допустимое

//для представления данных в столбце

};

struct Strip{//полоса распечатки таблицы

int nField;//число полей

int\* fieldWidth;//ширина полей в полосе (массив)

};

typedef map<string, void\*> Row;//строка таблицы

typedef map<string, ColumnDesc> Header;//заголовок таблицы

string GetTabNameFromPath(string path);

string ignoreBlanc(const string str);

void\* GetValue(string value, string columnName,Header hdr);

void\* SetValue(string value, string columnName,Header hdr);

bool comparator(DBType type,void \*obj1,Condition condition,void \*obj);

int GetLength(ColumnDesc colDesc);

//------------------- класс DBTableTxt ----------------------

class {

Header columnHeaders;

string tableName;

string primaryKey;

vector<Row> data;

string fileName;

public:

DBTableTxt(){}

DBTableTxt(string tabName);

DBTableTxt(string tabName,Header hdr,string primKey);

const char\* TypeName(DBType type);

~DBTableTxt(){}

vector<int> IndexOfRecord(void\* keyValue,string keyColumnName);

string valueToString(Row& row,string columnName);

void ReadDBTable(string fileName); //fileName=path+tableName

void PrintTable(int screenWidth);

void WriteDBTable(string fileName);//fileName=path+tableName

int GetSize();

DBType GetType(char\* columnName);

Row operator[](int ind);

void SetFileName(string path);

void SetTableName(string tName);

void SetPrimaryKey(string key);

string GetFileName();

string GetTableName();

string GetPrimaryKey(){return primaryKey;}

Header GetHeader();

void SetHeader(Header& hdr);

Row CreateRow();

Row GetRow(int index);

void AddRow(Row row,int index);

//Макет распечатки таблицы:

//screenWidth - ширина экрана (входной параметр)

//nStrip - число полос в распечатке (выходной параметр)

//strips[nStrip] - описание полос таблицы:   
 void CreateTableMaket(Strip\* &strips,int &nStrip,   
 int screenWidth);

friend void ReadDBTable1(DBTableTxt& tab,string fileName);   
 //fileName=path+tableName  
 friend void PrintDBTable1(DBTableTxt& tab,int screenSize);

};