ЛАБОРАТОРНАЯ РАБОТА №6.

Наследование.

Цель работы: Изучение языковых конструкций C++, реализующих механизм наследования классов; приобретение навыков разработки и использования производных классов.

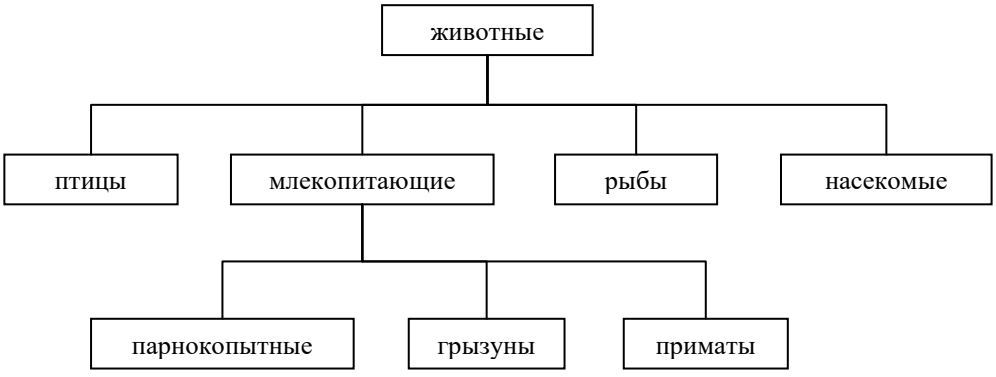
Теоретические сведения.

6.1) Отношение обобщения.

В лабораторной работе №4 нами была рассмотрена композиция классов как способ создания новых классов на основе уже существующих. Однако композиция (и агрегация) не являются единственным механизмом взаимодействия классов между собой. Другим активно используемым в ООП методом является *наследование классов* (class inheritance).

Известно, что композиция основана на отношении вида «целое-часть» (еще говорят об отношении «имеет…», «содержит…» или «has a…»). В отличие от композиции, при наследовании классы находятся между собой в отношении *обобщения*, которое выражают словами «является…» или «is a…». Например, класс `Животные` и класс `Млекопитающие` находятся между собой в отношении обобщения, так как любое млекопитающее является животным, то есть понятие животного является более общим, чем понятие млекопитающего. Аналогичное отношение можно установить между классами `Млекопитающие` и `Парнокопытные`, классами `Парнокопытные` и `Лошади` и т.д. Отношения обобщения позволяют, таким образом, строить сложные многоуровневые иерархии классов.

Рассмотрим теперь чуть подробнее, почему такого рода отношения между классами в ООП реализуются через механизм наследования. В качестве примера возьмем пару `Млекопитающие/Парнокопытные`, и рассмотрим отношения между соответствующими понятиями в природе. Действительно, любое животное отряда парнокопытных (лошадь, корова, жираф, бегемот и др.) обладает набором свойств, присущих всем млекопитающим (определенное строение тела, вскармливание детенышей молоком и т.д.). Можно говорить о том, что в природе парнокопытные *наследуют* часть своих свойств и поведения от млекопитающих. Эти свойства и поведение являются общими для всех млекопитающих. С другой стороны, парнокопытные имеют также характеристики, отличающие их от других млекопитающих, например, грызунов или приматов (см. схему).



Аналогичные отношения реализуются между программными классами в языке C++. Введем термины, используемые для описания наследственных отношений. Будем называть *базовым* класс, от которого производится наследование (в паре млекопитающие/парнокопытные – это класс млекопитающих). Иногда его называют родительским классом или классом-предком. Будем называть *производным* класс, образованный в результате наследования от родительского класса (в паре млекопитающие/парнокопытные – класс парнокопытных). Его также называют классом-наследником, классом-потомком или дочерним классом. Будем называть *иерархией наследования* все отношения между родительским классом и его потомками.

Идея наследования связана с тем, что образованный в результате новый класс получает от родителя (наследует) все его свойства и функциональность. При этом

- Класс-наследник имеет доступ к публичным и защищенным методам и полям класса родительского класса.
- Класс-наследник может добавлять свои данные и методы, а также переопределять методы базового класса.

6.2) Объявление производного класса.

Синтаксис объявления производного класса следующий

```
class имя_производного_класса : ключ_доступа имя_базового_класса
{
      // тело производного класса
};
```

В качестве ключа доступа здесь может использоваться одно из ключевых слов public, private и protected, которые реализуют различные виды наследования – открытое, закрытое и защищенное, соответственно. Вид наследования (public, private и protected) в данном случае определяет доступность полей и методов базового класса для объектов производного класса.

При наследовании класс-потомок получает в свое распоряжение все поля и методы, имеющиеся в родительском классе. Однако он может также *расширить* класс родителя, добавив в него новые поля и/или методы. Эти дополнительные элементы объявляются в теле производного класса.

Рассмотрим в качестве примера приложение «Книжный магазин», которое разрабатывается нами на протяжении всего лабораторного практикума. Мы имеем класс book для описания отдельной книги из книжного магазина, который содержит поля для хранения названия книги (title), ее авторов (authors), издательства (publisher), года выпуска (year) и количества страниц (pages). У нас также имеется класс bookstore, хранящий данные обо всех книгах в книжном магазине.

Предположим теперь, что ассортимент книжного магазина составляют не только оригинальные книги, написанные на русском языке, но и переводные издания. И пусть информация о переводной книге, среди прочего, содержит указание языка оригинала (английский, немецкий, японский и т.д.) и фамилию переводчика.

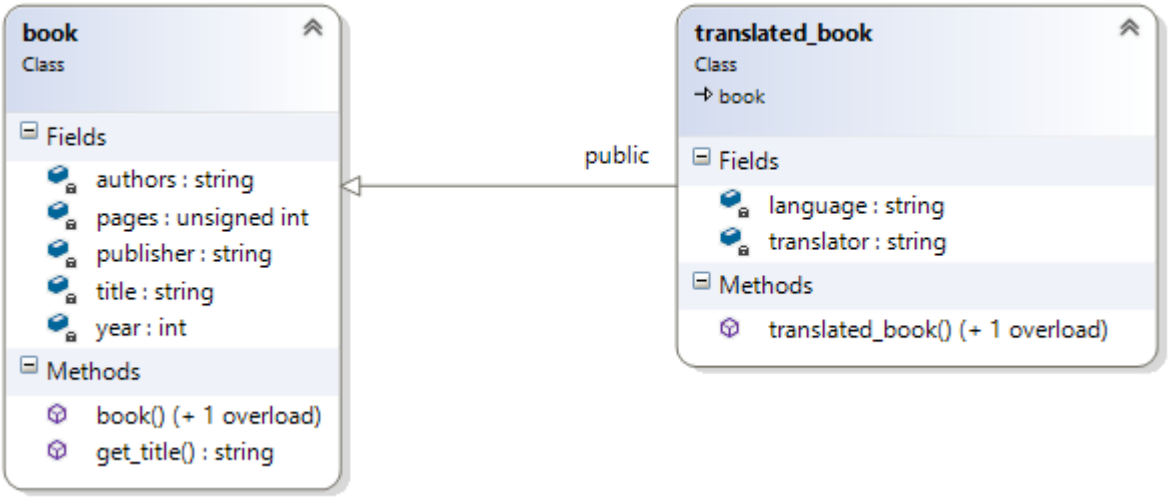
Разработаем новый класс переводной книги – translated_book. В данном случае между классами book и translated_book имеет место отношение обобщения – переводная

книга является частным случаем обыкновенной (оригинальной) книги. Поэтому класс translated_book будет реализован нами как наследник класса book

```
class translated_book : public book
{
private:
    string language;
    string translator;

public:
...
};
```

Обратим внимание на то, что в данном случае мы использовали открытое наследование (ключ доступа public). Это означает, что поля базового класса (title, authors и т.д.) будут иметь в производном классе тот же уровень доступа, как и в классе-родителе. Отметим также, что в производном классе мы объявили два дополнительных текстовых поля – это закрытые поля language (язык оригинала) и translator (переводчик). На UML-диаграмме классов наследование изображается стрелкой от производного класса к базовому (см. схему ниже). Над стрелкой указывается вид наследования.



6.3) Создание объекта производного класса. Доступ к элементам.

Создание объекта производного класса, то есть размещение экземпляра этого класса в компьютерной памяти, выполняется стандартным способом. Для этого необходимо указать имя производного класса и имя объекта, а также при необходимости указать параметры конструктора в круглых скобках.

```
translated_book abook(параметры_конструктора);
```

В случае, когда в классе определен конструктор по умолчанию, круглые скобки и параметры конструктора могут не указываться.

Для динамического размещения объекта используют оператор new

```
translated_book *pointer = new translated_book(параметры_конструктора);
```

Освобождение памяти в таком случае производится оператором delete

```
delete pointer;
```

Доступ к полям и методам производного класса осуществляется также обычным способом – с использованием оператора «точка» для статических объектов и оператора «стрелка» для указателей.

```
имя_объекта.поле_производного_класса = ...;
имя_объекта.метод_производного_класса(параметры);
указатель->поле_производного_класса = ...;
указатель->метод_производного_класса(параметры);
```

Здесь `имя_объекта` и `указатель` – имя и указатель на объект производного класса. Во всех указанных случаях применимы стандартные ограничения доступа: 1) закрытые элементы недоступны извне и из потомков, но доступны внутри класса; 2) защищенные элементы недоступны извне, но доступны из потомков и внутри класса; 3) открытые элементы доступны отовсюду.

Доступ к унаследованным элементам (т.е. полям и методам базового класса) осуществляется аналогичным образом

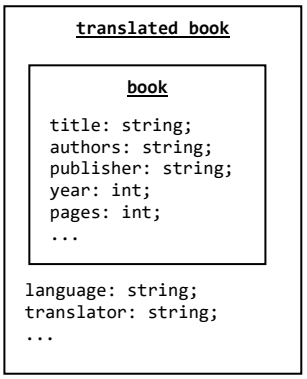
```
имя_объекта.поле_базового_класса = ...;
имя_объекта.метод_базового_класса(параметры);
указатель->поле_базового_класса = ...;
указатель->метод_базового_класса(параметры);
```

Следующий фрагмент кода демонстрирует инициализацию полей класса translated_book. Обратим внимание на то, что работа с собственными полями (language и translator) здесь ничем не отличается от работы с унаследованными полями (title, authors и т.д.).

```
translated_book book("", "", "", 0, 0, "", "");
book.title = "Эффективное использование C++";
book.authors = "Мейерс С.";
book.publisher = "ДМК Пресс";
book.year = 2014;
book.pages = 300;
book.language = "английский";
book.translator = "Мухин Н.А.";
```

Приведенный пример кода является корректным только для случая, если поля базового и производного классов объявлены открытыми (public). В противном случае попытка изменить значения любого поля будет приводить к ошибке времени компиляции.

6.4) Конструктор производного класса.



Объект производного класса можно рассматривать как объект в объекте: его «внутреннее ядро» составляет экземпляр базового класса, «внешнюю оболочку» – собственные поля и методы (см. рис. справа). Инициализация такого сложного объекта должна производиться в два этапа: сначала инициализируется ядро, затем – оболочка. При этом инициализацию ядра обеспечивает конструктор базового класса.

Рассмотрим возможную реализацию конструктора по умолчанию для класса translated_book

```
translated_book::translated_book() : book(), language(""), translator("")
{};
```

Обратим внимание на список инициализации этого конструктора: здесь сначала явным образом вызывается конструктор базового класса book, который «обнуляет» внутренние поля, а затем «обнуляются» собственные поля класса – language и translator. В результате получаем «пустой» объект, готовый к дальнейшему использованию.

Теперь рассмотрим конструктор переводной книги с аргументами. В качестве параметров мы будем передавать конструктору название книги, авторов, издательство, год выпуска, число страниц, язык оригинала и имя переводчика.

```
translated_book::translated_book(string ttl, string aut, string pbl, int yer,
                                 unsigned int pag, string lang, string tran) :
        book(ttl, aut, pbl, yer, pag), language(lang), translator(tran)
{
        cout << "\nВызван конструктор класса translated_book";
}
```

Как видно из приведенного примера, часть параметров конструктора производного класса передается напрямую конструктору базового класса. Другая часть параметров используется для инициализации собственных полей класса translated_book.

6.5) Переопределение методов базового класса в производном классе.

Любая функция базового класса может быть *переопределена* в производном классе. Переопределение в данном случае состоит в том, что мы объявляем в производном классе функцию с такой же сигнатурой, как и в базовом классе. В результате она «заслоняет» собой функцию базового класса, то есть реализует свой вариант метода, специфичный для производного класса.

Рассмотрим следующий пример. Пусть метод print() класса book выводит на экран информацию об оригинальной книге, то есть значения полей title, authors, publisher, year и pages. При открытом наследовании этот метод будет доступен и для класса-потомка translated_book. Однако в последнем случае функциональность метода недостаточна, так как он не выводит на экран значения дополнительных полей переводной книги: language и translator. Выход в данной ситуации может быть следующим: мы переопределяем метод print() для производного класса translated_book таким образом, чтобы он выводил на экран значения полей базового класса и собственных полей.

В качестве иллюстрации переопределим оператор вывода объекта translated_book в выходной поток

```
ostream& operator<<(ostream& stream, const translated_book& tbook)
{
   stream << static_cast<book>(tbook);
   stream << "Оригинал: " << tbook.language << endl;
   stream << "Перевод : " << tbook.translator << endl;
   return stream;
}
```

Здесь мы сначала вызываем оператор << базового класса book, который выводит на экран значения «внутренних» полей. Следом за этим на экран выводятся собственные поля производного класса. Функция static_cast здесь служит для преобразования типа translated_book в тип book позволяет вызвать метод базового класса явным образом.

6.6) Пример приложения Visual C++.

Расширим программный проект «Книжный магазин», добавив в ассортимент магазина переводные книги, изначально написанные на иностранных языках. Разработаем класс переводной книги translated_book как производный от имеющегося класса book. Для класса translated_book переопределим операторы потокового ввода/вывода, операцию проверки на эквивалентность.

```
======================== файл books.h ========================
```

```
#ifndef BOOKS_H
#define BOOKS_H

#include <string>
#include <iostream>
#include <fstream>
using namespace std;

class book
{
private:
	string title;
	string authors;
	string publisher;
	int year;
	unsigned int pages;
public:
	book() : title(""), authors(""), publisher(""), year(0), pages(0) {};
	book(string, string, string, int, unsigned int);
	string get_title();

	friend bool operator ==(book book1, book book2);
	friend ostream& operator<<(ostream& stream, const book& abook);
	friend istream& operator>>(istream& stream, book& abook);
	friend ofstream& operator<<(ofstream& stream, const book& abook);
	friend ifstream& operator>>(ifstream& stream, book& abook);
};

class translated_book : public book
{
private:
	string language;
	string translator;
public:
	translated_book() : book(), language(""), translator("") {};
	translated_book(string, string, string, int, unsigned int, string, string);

	friend bool operator ==(translated_book book1, translated_book book2);
	friend ostream& operator<<(ostream& stream, const translated_book& abook);
	friend istream& operator>>(istream& stream, translated_book& abook);
	friend ofstream& operator<<(ofstream& stream, const translated_book& abook);
	friend ifstream& operator>>(ifstream& stream, translated_book& abook);
};

#endif
```

========== файл books.cpp ==========

```cpp
#include "stdafx.h"
#include "books.h"

using namespace std;

// Методы базового класса book

book::book(string ttl, string aut, string pbl, int yer, unsigned int pag)
{
	title = ttl;
	authors = aut;
	publisher = pbl;
	year = yer;
	pages = pag;
	cout << "\nВызван конструктор класса book";
}

string book::get_title()
{
	return title;
}

bool operator ==(book book1, book book2)
{
	if (book1.title != book2.title) return false;
	if (book1.authors != book2.authors) return false;
	if (book1.publisher != book2.publisher) return false;
	if (book1.year != book2.year) return false;
	if (book1.pages != book2.pages) return false;
	return true;
}

ostream& operator<<(ostream& stream, const book& abook)
{
	stream << "Название: " << abook.title << endl;
	stream << "Автор(ы): " << abook.authors << endl;
	stream << "Издатель: " << abook.publisher << endl;
	stream << "Год     : " << abook.year << endl;
	stream << "Страниц : " << abook.pages << endl;
	return stream;
}

istream& operator>>(istream& stream, book& abook)
{
	getline(stream, abook.title);
	getline(stream, abook.authors);
	getline(stream, abook.publisher);
	stream >> abook.year;
	stream >> abook.pages;
	stream.get();
	return stream;
}

ofstream& operator<<(ofstream& stream, const book& abook)
{
	stream << abook.title << endl;
	stream << abook.authors << endl;
	stream << abook.publisher << endl;
	stream << abook.year << endl;
	stream << abook.pages << endl;
	return stream;
}
```

```
ifstream& operator>>(ifstream& stream, book& abook)
{
	getline(stream, abook.title);
	getline(stream, abook.authors);
	getline(stream, abook.publisher);
	stream >> abook.year;
	stream >> abook.pages;
	stream.get();
	return stream;
}

// Методы производного класса translated_book

translated_book::translated_book(string ttl, string aut, string pbl, int yer, unsigned
int pag, string lang, string tran) :
book(ttl, aut, pbl, yer, pag), language(lang), translator(tran)
{
	cout << "\nВызван конструктор класса translated_book";
}

bool operator ==(translated_book book1, translated_book book2)
{
	bool are_equal_books = static_cast<book>(book1) == static_cast<book>(book2);
	bool are_equal_langs = book1.language == book2.language;
	bool are_equal_trans = book1.translator == book2.translator;
	if (are_equal_books && are_equal_langs && are_equal_trans)
		return true;
	return false;
}


ostream& operator<<(ostream& stream, const translated_book& tbook)
{
	stream << static_cast<book>(tbook);
	stream << "Оригинал: " << tbook.language << endl;
	stream << "Перевод : " << tbook.translator << endl;
	return stream;
}

istream& operator>>(istream& stream, translated_book& tbook)
{
	stream >> static_cast<book>(tbook);
	getline(stream, tbook.language);
	getline(stream, tbook.translator);
	return stream;
}

ofstream& operator<<(ofstream& stream, const translated_book& tbook)
{
	stream << static_cast<book>(tbook) << endl;
	stream << tbook.language << endl;
	stream << tbook.translator << endl;
	return stream;
}

ifstream& operator>>(ifstream& stream, translated_book& tbook)
{
	stream >> static_cast<book>(tbook);
	getline(stream, tbook.language);
	getline(stream, tbook.translator);
	return stream;
}
```

Расширим также класс-контейнер bookstore, определив в нем два независимых массива: один для оригинальных русскоязычных книг (объектов класса book), другой – для переводных книг (объектов класса translated_book). Функции файлового ввода/вывода изменятся соответствующим образом (см. код ниже).

```
====================== файл bookstore.h ======================
```

```
#ifndef BOOKSTORE_H
#define BOOKSTORE_H

#include <string>
#include <fstream>
#include "books.h"

class bookstore
{
private:
	int max_num_obooks;
	int max_num_tbooks;
	int num_obooks;
	int num_tbooks;
	book *obooks;                // массив оригинальных (непереводных) книг
	translated_book *tbooks;     // массив переводных книг

public:
	bookstore(int max_ob, int max_tb);
	~bookstore();

	void operator +=(const book &obook);
	void operator +=(const translated_book &tbook);

	friend ostream& operator<<(ostream& stream, const bookstore& astore);

	void read_obooks_from_file(string filename);
	void read_tbooks_from_file(string filename);
	void write_obooks_to_file(string filename);
	void write_tbooks_to_file(string filename);
	void display_all();
	void find_book(string atitle);
};

#endif
```

```
====================== файл bookstore.cpp ======================
```

```
#include "stdafx.h"
#include <iostream>
#include "bookstore.h"

using namespace std;

bookstore::bookstore(int max_ob, int max_tb)
{
	max_num_obooks = max_ob;
	max_num_tbooks = max_tb;
	obooks = new book[max_num_obooks];
	tbooks = new translated_book[max_num_tbooks];
	num_obooks = 0;
	num_tbooks = 0;
	cout << "\nВызван конструктор класса bookstore:";
	cout << "\n    выделено объектов (для оригинальных изданий) - " << max_num_obooks;
	cout << "\n    выделено объектов (для переводных изданий) - " << max_num_tbooks;
```

```
	cout << "\n    загружено книг (оригинальных) - " << num_obooks;
	cout << "\n    загружено книг (перводных) - " << num_tbooks;
	cout << endl;
}

bookstore::~bookstore()
{
	cout << "\nВызван деструктор класса bookstore:";
	max_num_obooks = 0;
	delete[] obooks;
	num_obooks = 0;

	max_num_tbooks = 0;
	delete[] tbooks;
	num_tbooks = 0;
	cout << "\n    выделенная память освобождена";
}

void bookstore::operator+=(const book &obook)
{
	if (num_obooks < max_num_obooks)
	{
		obooks[num_obooks] = obook;
		num_obooks++;
	}
}

void bookstore::operator+=(const translated_book &tbook)
{
	if (num_tbooks < max_num_tbooks)
	{
		tbooks[num_tbooks] = tbook;
		num_tbooks++;
	}
}

ostream& operator<<(ostream& stream, const bookstore& astore)
{
	stream << "\n\n\n ВЕСЬ АССОРТИМЕНТ КНИЖНОГО МАГАЗИНА:\n";
	stream << "\n    А) Оригинальные издания\n";
	for (int i = 0; i < astore.num_obooks; i++)
	{
		stream << "------------------------------\n";
		stream << astore.obooks[i] << endl;
	}
	stream << "\n    Б) Переводные издания\n";
	for (int i = 0; i < astore.num_tbooks; i++)
	{
		stream << "------------------------------\n";
		stream << astore.tbooks[i] << endl;
	}
	return stream;
}


void bookstore::read_obooks_from_file(string filename)
{
	ifstream infile;
	infile.open(filename);
	if (!infile.is_open())
	{
		cout << "\n\nФайл данных не найден!\n";
		system("pause");
		exit(1);
	}
```

```
	int N;
	infile >> N;
	infile.get();

	for (int i = 0; i < N; i++)
	{
		book new_book;
		infile >> new_book;
		this->operator+=(new_book);
	}
	infile.close();

	cout << "\nЗагружены оригинальные книги из файла " << filename << ":";
	cout << "\n    число загруженных книг - " << num_obooks << endl;
}

void bookstore::write_obooks_to_file(string filename)
{
	ofstream outfile;
	outfile.open(filename);
	outfile << num_obooks << endl;
	for (int i = 0; i < num_obooks; i++)
		outfile << obooks[i];
	outfile.close();

	cout << "\nДанные записаны в файл " << filename << ":";
	cout << "\n    записано оригинальных книг - " << num_obooks;
}

void bookstore::write_tbooks_to_file(string filename)
{
	ofstream outfile;
	outfile.open(filename);
	outfile << num_tbooks << endl;
	for (int i = 0; i < num_tbooks; i++)
		outfile << tbooks[i];
	outfile.close();

	cout << "\nДанные записаны в файл " << filename << ":";
	cout << "\n    записано переводных книг - " << num_tbooks;
}

void bookstore::find_book(string atitle)
{
	cout << "\n\nИщем книгу с названим \"" << atitle << "\"";
	bool found = false;
	for (int i = 0; i < num_obooks; i++)
	{
		if (obooks[i].get_title() == atitle)
		{
			found = true;
			cout << "\nНайдена книга:\n";
			cout << obooks[i];
		}
	}

	for (int i = 0; i < num_tbooks; i++)
	{
		if (tbooks[i].get_title() == atitle)
		{
			found = true;
			cout << "\nНайдена книга:\n";
			cout << tbooks[i];
		}
```

```
	}
	if (!found)
		cout << "\nКнига с таким названием не найдена!\n";
}
```

`================` основной модуль – lab6.cpp `=================`

```
#include "stdafx.h"
#include <string>
#include "bookstore.h"
#include "books.h"

void main()
{
	setlocale(LC_ALL, "rus");

	bookstore my_store(25, 10);

	book book1("Дети капитана Гранта", "Верн Ж.", "Эксмо", 2003, 800);
	my_store += book1;

	translated_book book2("Эффективное использование C++", "Мейерс С.",
				"ДМК Пресс", 2014, 300, "английский", "Мухин Н.А.");
	my_store += book2;

	cout << my_store;
	cout << "\n\n";
	system("pause");
}
```

`================================================================`

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ №6.

1) Разработайте производный класс как наследник базового класса для своего варианта задания. Добавьте в производный класс несколько дополнительных полей. Определите конструктор по умолчанию и конструктор с параметрами для производного класса. Переопределите соответствующие методы в производном классе.
2) Реализуйте хранение объектов базового и производного классов в контейнере. Измените методы класса-контейнера соответствующим образом: реализуйте запись и чтение данных из файла, поиск нужного объекта в коллекции и др.
3) Продемонстрируйте работу механизмов наследования и переопределения функций в своей программе.

Содержание отчета по лабораторной работе №6.

1) Стандартная «шапка» отчета
2) Цель: формулировка цели работы
3) Теория: краткие сведения о реализации наследования с примерами (объем 2-3 стр.)
4) Диаграмму классов программы.
5) Программный код
   - объявление производного класса,
   - определение конструктора по умолчанию и конструктора с параметрами для производного класса,
   - переопределение одного из методов производного класса.