15. Объектно-ориентированные методы анализа и проектирования ПО. Виртуальные функции и абстрактные классы, перегрузка функций и методов. Шаблоны классов.

Кратко о объектно-ориентированном проектировании (пункты 1.3-1.4), подробнее пункт 2.1 (в конце), совсем подробно глава 6.
http://www.helloworld.ru/texts/comp/other/oop/index.htm

Виртуальные функции, абстрактные классы
http://aco.ifmo.ru/el_books/applied_programming/lectures/part5-2.html

Перегрузка функций
http://aco.ifmo.ru/el_books/applied_programming/lectures/part6-1.html#6_1_1

Шаблоны классов
http://aco.ifmo.ru/el_books/applied_programming/lectures/part6-3.html#6_3_3

Методы анализа и проектирования ПО:
http://citforum.ru/programming/application/program/1.shtml

Объектно-ориентированное программирование - это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

В данном определении можно выделить три части: 1) ООР использует в качестве базовых элементов *объекты,* а не алгоритмы (иерархия "быть частью"); 2) каждый объект является *экземпляром* какого-либо определенного *класса;* 3) классы организованы *иерархически* ("is a"). Программа будет объектно-ориентированной только при соблюдении всех трех указанных требований. В частности, программирование, не основанное на иерархических отношениях, не относится к ООР, а называется *программированием на основе абстрактных типов данных.*

В соответствии с этим определением не все языки программирования являются объектно-ориентированными. Страуструп определил так: "если термин *объектно-ориентированный язык* вообще что-либо означает, то он должен означать язык, имеющий средства хорошей поддержки объектно-ориентированного стиля программирования... Обеспечение такого стиля в свою очередь означает, что в языке удобно пользоваться этим стилем. Если написание программ в стиле ООР требует специальных усилий или оно невозможно совсем, то этот язык не отвечает требованиям ООР". Теоретически возможна имитация объектно-ориентированного программирования на обычных языках, таких, как Pascal и даже COBOL или ассемблер, но это крайне затруднительно. Карделли и Вегнер говорят, что: "язык программирования является объектно-ориентированным тогда и только тогда, когда выполняются следующие условия:

- Поддерживаются объекты, то есть абстракции данных, имеющие интерфейс в виде именованных операций и собственные данные, с ограничением доступа к ним.
- Объекты относятся к соответствующим типам (классам).
- Типы (классы) могут наследовать атрибуты супертипов (суперклассов)".

Поддержка наследования в таких языках означает возможность установления отношения "is-a" ("есть", "это есть", " - это"), например, красная роза - это цветок, а цветок - это растение. Языки, не имеющие таких механизмов, нельзя отнести к объектно-ориентированным. Карделли и Вегнер назвали такие языки *объектными,* но не *объектно-ориентированными.* Согласно этому определению объектно-ориентированными языками являются Smalltalk, Object Pascal, C++ и CLOS, а Ada - объектный язык. Но, поскольку объекты и классы являются элементами обеих групп языков, желательно использовать и в тех, и в других методы объектно-ориентированного проектирования.

Объектно-ориентированное проектирование. Программирование прежде всего подразумевает правильное и эффективное использование механизмов конкретных языков программирования. Проектирование, напротив, основное внимание уделяет правильному и эффективному структурированию сложных систем. Мы определяем объектно-ориентированное проектирование следующим образом:

Объектно-ориентированное проектирование - это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

В данном определении содержатся две важные части: объектно-ориентированное проектирование 1) основывается на объектно-ориентированной декомпозиции; 2) использует многообразие приемов представления моделей, отражающих логическую (классы и объекты) и физическую (модули и процессы) структуру системы, а также ее статические и динамические аспекты.

Именно объектно-ориентированная декомпозиция отличает объектно-ориентированное проектирование от структурного; в первом случае логическая структура системы отражается абстракциями в виде классов и объектов, во втором - алгоритмами. Иногда мы будем использовать аббревиатуру *OOD,* object-oriented design, для обозначения метода объектно-ориентированного проектирования, изложенного в этой книге.

Объектно-ориентированный анализ (или *OOA, object-oriented analysis*) направлен на создание моделей реальной действительности на основе объектно-ориентированного мировоззрения.

Объектно-ориентированный анализ - это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.

Как соотносятся OOA, OOD и OOP? На результатах OOA формируются модели, на которых основывается OOD; OOD в свою очередь создает фундамент для окончательной реализации системы с использованием методологии OOP.

Виртуальные функции – функции базового класса, которые можно заместить в каждом производном классе. Если базовый класс вызывает перегруженную виртуальную функцию, то всегда будет вызываться функция наследника.

Для того, чтобы сделать функцию виртуальной, при ее описании в базовом классе необходимо указать ключевое слово virtual

```
virtual void print() const;
//////////////////////////////////////////
void main ()
{
   // виртуальные функции важны, когда создаются указатели на класс
         Detail *p=new Lens;
}
//////////////////////////////////////////
```

Использование виртуальных функций важно, когда создается экземпляр указателя на класс-наследник, если при этом экземпляр описан как базовый класс, но создается как наследник.

```
Detail *p=new Lens;
```

Делать виртуальными можно практически все функции, за исключением конструкторов и оператора =. Особый интерес представляют виртуальные деструкторы. Виртуальным деструктор делают, если для правильного освобождения памяти необходимо, чтобы деструктор всегда вызывался для класса-наследника.

Абстрактные классы

Некоторые базовые классы (например, класс Detail) представляют собой абстрактную концепцию, для которой не могут существовать экземпляры. Невозможно нарисовать абстрактную деталь или выполнить расчет прохождения луча через деталь.

В таких случаях базовый класс делают **абстрактным**, то есть классом, экземпляр которого создать нельзя, но можно создать экземпляры его наследников.

Абстрактным называется класс, имеющий чисто виртуальные функции. **Чисто виртуальная функция** – это функция, которая не определена в базовом классе, и обязательно должна быть перегружена в классах наследниках. Если какая-то абстрактная функция не будет перегружена в классе наследнике – компилятор выдаст сообщение об ошибке.

В результате функции базового класса могут безопасно вызывать любые свои функции, в том числе и виртуальные, потому что они гарантированно будут перегружены в классах наследниках.

Перегрузка функций

Как правило, разным функциям дают различные имена, но когда функции выполняют одинаковые действия, но для объектов разных типов, может оказаться удобным присвоить одно и то же имя.

Использование одного имени для функций, выполняющих действия с аргументами разных типов, называется **перегрузкой.**

функции, выполняющие одинаковые действия для разных типов:	перегруженные функции:

print_double(double d);
print_Lens(Lens l);
print_2(double x, double y);
	void print(double d);
	void print(Lens l);
	void print(double x, double y);

Все эти функции имеют одинаковое имя, но разный набор параметров, поэтому компилятор легко отличит их друг от друга при вызове по набору аргументов. Процесс поиска подходящей функции из набора перегруженных осуществляется компилятором по следующим правилам:

1. Проверка точного соответствия типов аргументов функции и передаваемых параметров
2. Попытка “повышения типов” (short -> int, flot -> double и т.д.)
3. Попытка стандартных преобразований (int -> double, double -> int, указатели -> void* и т.д.)
4. Преобразование, явно задаваемое программистом

Возвращаемые типы не участвуют в определении какую из перегруженных функций вызвать.

Параметрический полиморфизм - еще одно из проявлений принципа полиморфизма, которое позволяет многократно использовать один и тот же код применительно к разным типам. Реализация функции или класса осуществляется один раз, но при этом тип указывается как параметр этой функции или класса. При использовании этой функции или класса параметра типа заменяется на встроенный или абстрактный тип данных и универсальных код, не зависящий от типа, работает с указанным типом. Существуют целые библиотеки таких функций и классов (например, STL) и такая технология называется обобщенное программирование.

Шаблоны (templates) – средство для реализаций параметризированных классов и функций на языке C++.

Шаблоны позволяют реализуют в языке C++ такое отношение между классами, как инстанцирование. При создании шаблона класса тип одного или нескольких его переменных

членов задаются в качестве параметра. При использовании этого шаблона необходимо будет указать, какой тип использовать в качестве параметра. Это и называется инстанцированием. Например, обобщенный класс Complex. Многие классы, которые используются для хранения и управления гомогенными структурами данных, реализуют в виде шаблонов. И действительно, независимо от типов хранимых в нем элементов, класс Complex должен выполнять одни и те же функции

Процесс генерации объявления класса по шаблону и аргументу называется инстанцированием шаблона.