ООП Макеева. Шпаргалка

- **Переменная** – область в памяти, имеющая имя, значение и тип

- **Память** – нумерованная последовательность числовых двоичных байтов

- **Адрес переменной** – номер первого байта, занимаемого переменной в памяти (для каждой переменной можно узнать ее адрес, с помощью операции &)

- **Указатель** – (int \*r = &a) переменная, содержащая адрес другой переменной

- **Ссылка** – (int &r = a) другое имя для той же самой переменной (~~ссылки это указатели которые автоматически разыменовываются~~). Принципиальная разница с указателями – ссылки не имеют собственного места в памяти и ссылки нельзя переназначить.

-

- **Композиция** – это отношение “состоит из…”, это гибкий способ связывать классы

- **Наследование** – это отношение “является частным случаем…” или “ведет себя как…”, это жесткий способ связывать классы, но с ней проще работать с функциональностью чужого класса (c.x в отличии от c.point.x)

**Их объединяет технологическое сходство** – получить функциональность одного класса внутри другого.

**Различие** – наследование связывает классы во время компиляции, композиция может связывать классы во время выполнения.

**Правило, что где использовать** – если класс Б должен являться частным случаем или вариацией класса А, (а значит интерфейс Б должен совпадать с интерфейсом А), то лучше использовать наследование. Если же классу Б нужна лишь часть функциональности от А, то лучше использовать композицию/агрегацию.

**- Урезание** – урезание от объектов свойств при помещении дочернего объекта в родительский

✎ Это позволяет использовать один и тот же контейнер для хранения объектов разных классов. Так вместо отдельных списков для парт, досок, стульев, шкафов и другой мебели и необходимости при появлении нового типа мебели создавать ещё один список, можно использовать один единый список мебели, в котором можно хранить сразу всё. В том числе и те подклассы мебели, которые ещё не появились, но, возможно, появятся позже.

✎ Это позволяет коду работать с теми объектами, которые даже не были ещё созданы тогда, когда этот код писался, а значит позволяет использовать без изменений старый код с новыми объектами

Desk d;

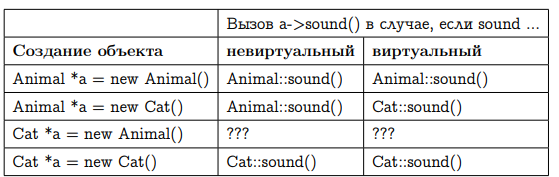
Furniture f

f = d;

f.student\_capacity;

- **Виртуальные методы** – они нужны когда объекты потомки будут лежать в переменных классов-предков и мы хотим со всеми ними работать единообразно, не зная какому конкретно классу-потомку они принадлежат, а просто вызывая при этом общие для всех них методы, определенные в классе-предке. При этом чтобы при вызове этого общего для каждого объекта потомка метода вызывался метод свой специфичный.

**Различие виртуального от не виртуального** – когда объект класса-потомка лежит в переменной класса-предка и у предка вызывается виртуальный метод, перекрытый в потомке



- **Полиморфизм (динамическая диспетчеризация)** – когда мы хотим работать с разными объектами единообразно, через единый интерфейс, но при этом ожидать различного поведения в соответствии с конкретным типом объекта. С точки зрения компилятора принять решение какой код будет вызван в строчке вида x->foo() при foo виртуальной можно будет только динамически в рантайме тк это зависит от реального типа объекта x.

Удобства – объявляя деструктор виртуальным методом, мы можем избежать ошибок утечки памяти при

Base \*x = new Desc ( ) ;

x−>delete( );

когда у нас вызывается деструктор от объекта класса предка, а не потомка.

**Правило при хранении объектов класса-потомка в объектах класса-предка** - виртуальный деструктор в базовом классе обязательно нужен тогда, когда вы создаёте объекты классов-потомков и храните их в переменных классов-предков. В этом случае при удалении объекта-потомка, лежащего в переменной класса-предка, вызовется полная корректная последовательность деструкторов, а не только деструктор класса-предка. Вызов виртуального метода «проваливается» до самого глубокого перекрытого метода из тех, которыми обладает вызываемый объект, не только если мы вызываем метод, обращаясь к переменной. То же самое верно и для вызова метода изнутри другого метода

Абстрактный метод – не имеющий реализации вовсе.

Класс, содержащий хотя бы один абстрактный метод, абстрактный класс.

Если мы унаследуемся от абстрактного класс и не перекроем абстрактный метод, то не сможем скомпилировать программу.

Общее правило: виртуальный деструктор в базовом классе обязательно нужен тогда, когда вы создаёте объекты классов-потомков и храните их в переменных классов-предков. В этом случае при удалении объекта-потомка, лежащего в переменной класса-предка, вызовется полная корректная последовательность деструкторов, а не только деструктор класса-предка

**Мы не должны запрашивать у объекта информацию и принимать какое-то решение за него. Мы должны наоборот, дать ему достаточно информации для того, чтобы он сам смог принять решение**