**ЛЕКЦИЯ 7**

**ПЕРЕГРУЗКА ОПЕРАТОРОВ И НАСЛЕДОВАНИЕ**

**ДРУЖЕСТВЕННЫЕ ФУНКЦИИ (ДФ)**

1. Друж. функции не является компонентом класса, имеет доступ к защищенным и собственным компонентам класса, определена в классе со спецификатором friend;
2. Дружественная функция при вызове не использует this;
3. Объекты класса должны передаваться дф только через параметры;
4. Итого: дружественна функция не может быть компонентом класса, к которому является другом. Может быть глобальной или компонентом другого класса, который определен раньше.

**КЛАССЫ ДРУЗЬЯ (КД)**

1. Дружественный класс должен быть (если он друг) другом другому классу;
2. Все методы дк являются дружественными другому классу;
3. Дк должен быть определен вне тела класса, который предоставляет дружбу;

**ПЕРЕГРУЗКА ОПЕРАТОРОВ**

1. Есть множество операций, которые применены к данным определенного типа. Нам надо сделать так, чтобы операторы применялись корректно к классам. Решение вопроса – перегрузка операторов;
2. Перегрузка операторов позволяет определять действия, которые выполняет оператор;
3. Перегрузка – создание функции, создание которой содержит слово **operator** и символ оператора (**+**, **-**, **==** и т.д);
4. Может быть определена как член класса, внешняя функция, как дружественная функция;

**ИСПОЛЬЗОВАНИЕ ПАРАМЕТРОВ**

1. Операции бывают унарные (с одним) и бинарные (с двумя);
2. Могут быть определены способами:

А) Компонентная функция без параметров (с одним);

Б) Глобальная с одним (двумя) параметрами;

**УНАРНЫЕ ОПЕРАТОРЫ**

1. В постфиксных операторах должен быть один параметр (int). Он ***АБСОЛЮТНО БЕСПОЛЕЗЕН, НО БЫТЬ ОБЯЗАН***;

**БИНАРНЫЕ ОПЕРАТОРЫ**

1. Для перегрузки необходимо определить приоритет и ассоциативность;
2. Приоритет – порядок операции в выражении если операций несколько;
3. Ассоциативность – порядок выполнения нескольких операторов с одинаковым приоритетом (слева или справа);

**ПЕРЕГРУЗКА ОПЕРАТОРА ПРИСВАИВАНИЯ**

1. Оператор не наследуется;
2. Определен по умолчанию для каждого класса определен как побитовое копирование значений другого класса (включая его адрес);
3. Может быть перегружен только как функция – компонент класса;
4. Стоит перегружать, когда класс содержит динамически инициализированные элементы, чтобы избежать утечку памяти.
5. ***Разобрать работу 10!!! (к 8 апреля);***
6. C = B = A (сначала B = A, потом C = B). Ассоциация идет слева направо;

**ПЕРЕГРУЗКА ОПЕРАТОРА ИНДЕКСАЦИИ**

1. a[“x”] = 12; b[“y”] = 15. Альтернативный способ задавать значения элементам.

**НАСЛЕДОВАНИЕ КЛАССОВ С++**

1. Наследование – механизм описания нового класса (потомка, производного, наследника, дочкой, сыночком) на основе существующего класса (родителя, базового класса, суперкласса);
2. Следует использовать когда: классы связаны смыслово через методы и параметры (машина -> двигатель -> тормоз). Должна четко просматриваться иерархия;
3. Синтаксис: **class \_имя\_ : спецификатор (public, private, protected) \_имя\_родителя\_**;

**СПЕЦИФИКАТОРЫ ДОСТУПА В НАСЛЕДОВАНИИ**

1. Public – в дочь передадутся публичные и защищенные элементы, не изменят свои модификаторы доступа (публичные останутся публичными, защищенные останутся защищенными);
2. Protected – аналогично public, однако публичные элементы родителя становится защищенными (приватные все еще не передаются, защищенные родителя становятся защищенными ребенка);
3. Private – все родительские элементы становятся приватными элементами дочери (но приватные элементы родителя дочь все равно не получает);
4. Если делать наследование от класса без спецификатора, то по умолчанию спецификатор – private. Если идет наследование от структуры, то по умолчанию будет public (наследуем в класс);

**ПЕРЕОПРЕДЕЛЕНИЕ СПЕЦИФИКАТОРОВ ВНУТРИ ДОЧЕРНЕГО КЛАССА**

1. Синтаксис:

**новый\_тип\_доступа:**

***using* имя\_базового\_класса::имя\_поля\_;**

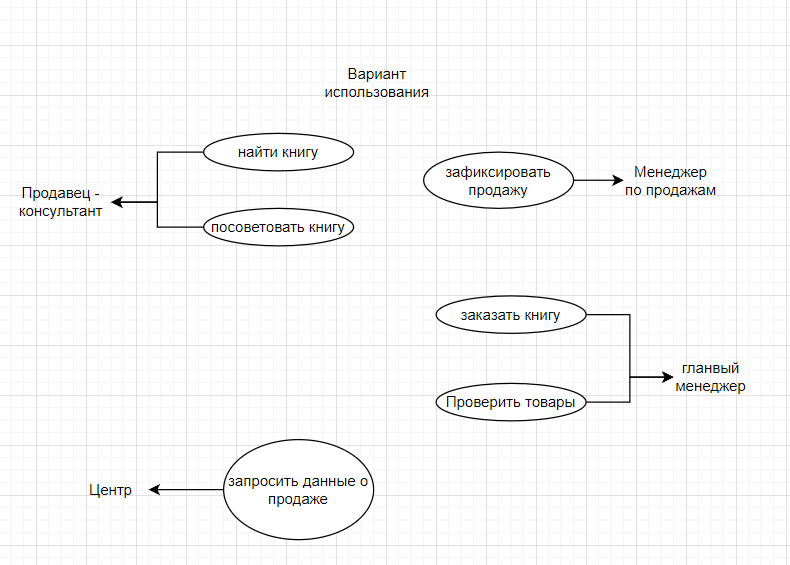
1. Нельзя переопределять так private поля родителя, так как они не передаются в дочь;

**КОНСТРУКТОРЫ И ДЕЗРУКТОРЫ ПРИ НАСЛЕДОВАНИИ**

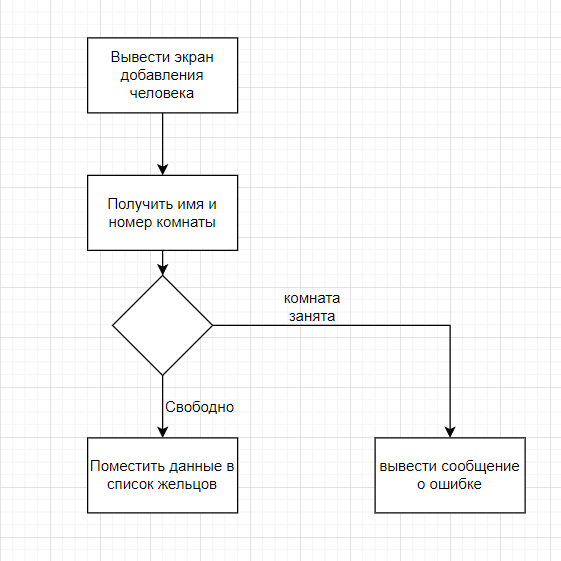
1. Обращение дочерних классов к конструкторам обращается по принципу **изнутри – наружу**. Вызов конструкторов работает по принципу “снаружи во внутрь”;
2. Дезрукторы работают по аналогичному принципу, вызов работает по принципу “**изнутри наружу**”;
3. Конструкторы и дезрукторы не наследуются;
4. Конструкторы вызываются начиная с базового класса, заканчивая самым последним дочерним;
5. Дезрукторы наоборот (от последнего до первого);
6. Так как конструкторы не наследуются, следует инициализировать элементы через конструктор родительского класса;
7. Если конструктор и дезруктор не указать, то вызовутся дефолтные, что может привести к утечке памяти;
8. Всем наследованным и текущим полям класса можно задать значения;
9. Имеется возможность четко указать, к какому конструктора родителя должна обращаться дочь. Синтаксис: **дочь():родитель(\_параметры\_конструктора) {}**; В противном случае компилятор всегда будет обращаться к конструктору по умолчанию.

**UML ДИАГРАММЫ**

1. Использовать с 3-ей лабораторной работы;
2. UML – юнифицированны язык моделирования, для визуализации, проектирования, документирования программных систем;
3. Артефакт – физическая сущность, которая смыслово представляет собой какую – либо часть рассматриваемого проекта;
4. Диаграмма вариантов: последовательность событий, действий с артефактами, которые приводят к результату;
5. Описание элемента в крупных шагах.
6. Пример:



1. Диаграмма действий UML:



**СУЩНОСТИ. ВИДЫ**

1. Сущности – основные элементы объекто ориентированного языка. С их помощью можно создать конкретные модели;
2. Виды сущностей: структурные, поведенческие, группирующие, аннотационные.

**СТРУКТУРНЫЕ СУЩНОСТИ**

1. Статические сущности UML модели;
2. Объект – сущность, обладающая уникальностью и инкапсулирует в себе состояние и поведение;
3. Класс – множество сущностей (объектов) с общими атрибутами. Определено также состояние через атрибуты и поведение через операции;
4. Интерфейс – множество объединенных именем (смыслом) операций, определяют набор действий, которые запрошены потребителем, а предоставлены поставщиком;
5. Узел – преднозначен для вычислительных процессов, например с артефактами;
6. Кооперация – объекты, которые связаны единой целью;
7. Действующее лицо – сущность, которая взаимодействует с системой и находится вне её;
8. Компонент – часть системы (модуль) с определенным набором интерфейсов;
9. Артефакт – элемент информации, который используется и порождается в системе;

**ПОВЕДЕНЧЕСКИЕ СУЩНОСТИ – ДИНАМИЧЕСКИЕ ЧАСТИ UML**

1. Состояние – период жизненного цикла объекта: осуществляется деятельность либо ожидание;
2. Деятельность – состояние, которое называется ***не атомарным***: продолжительно по времени, не дает гарантий выполнения вычислений, могут быть прерваны в любой момент;
3. Действие – атомарное вычисление, однократное или не выполненое;

**ГРУППИРУЮЩИЕ СУЩНОСТИ**

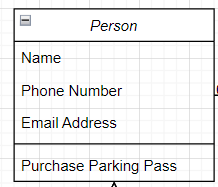
1. Пакет – группа элементов модели (в том числе пакетов);

**АННОТАЦИОННЫЕ СУЩНОСТИ**

1. Поясняющие части UML модели – комментарии (примечания), которые описывают что – то в модели;

**КЛАСС, КАК СТРУТУРНАЯ СУЩНОСТЬ**

1. Графическое изображение класс (пример):



В верхнем блоке поля, в нижнем методы;

1. - - обозначает private, # - protected, + - public;

**АТРИБУТЫ**

1. Синтаксис: **видимость имя кратность : тип = начальное значение (свойства)**;
2. Существование кратности означает, что атрибут - массив;
3. Свойства – дополнительные сведения, ограничения или значения;
4. Значения свойств изменяемого атрибута:

**- {changeable} – можно менять без изменений;**

**- {addOnly} – для массивов, можно только добавлять, запоминает историю изменений;**

**- {frozen}, {readOnly} – как const**;

1. Синтаксис метода: **видимость имя (параметры) : тип {свойства}**;
2. Ключевые слова для передачи параметров:

**- in – взодной;**

**- out – выходной;**

**- inout – можно ввести, а потом вывести;**

**- return – возвращаемое значение.**

1. Примеры:

**- move() – самый минимум;**

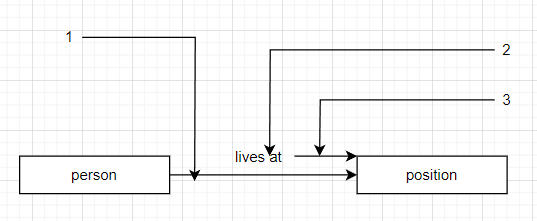
**- +move(in from, in to);**

**- +move(in in from:Department, in to:Department);**

1. Шаблоны: **имя\_классификатора : имя\_шаблона <аргументы>**;

**ОТНОШЕНИЕ МЕЖДУ КЛАССАМИ**

1. Ассоциации: когда объекты одного класса связаны с объектами другого класса таким образом, что можно двигаться от объектов одного класса к другому;

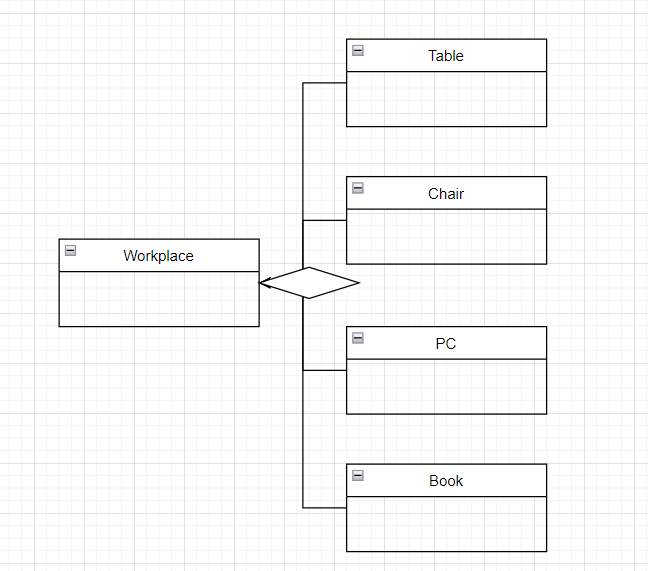


Где: 1 – отношение между объектами, 2 – глагол, объясняющий суть ассоциации, 3 – как читать (слево направо или наоборот);

1. Кратность – число объектов, принимающих участие в отношении. Может быть числом, диапозоном и звездочкой (невозможно определить точное количество). В примере выше – одна позиция на одного человека;

**АГРЕГАЦИЯ**

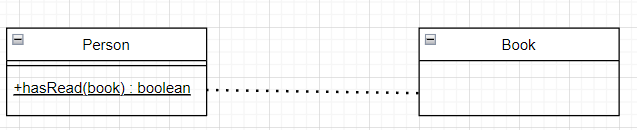
1. Один класс – часть другого класса, но могут существовать и по отдельности:

****

1. Композиция – разновидность агрегации, когда составляющие объект классы исчезают вместе с главным (отличие – ромб закрашен);

**ЗАВИСИМОСТЬ**

1. Изменение одного элемента обозначает изменение другого:



Книга зависит от человека

**РЕАЛИЗАЦИЯ**

1. Отношение интерфейса и объектов, реализующий данный интерфейс:

