Лекция 10

Как строить модель?

В структурном программировании были этапы разработки:

1. Анализ.
2. Проектирование.
3. Кодирование.
4. Тестирование.

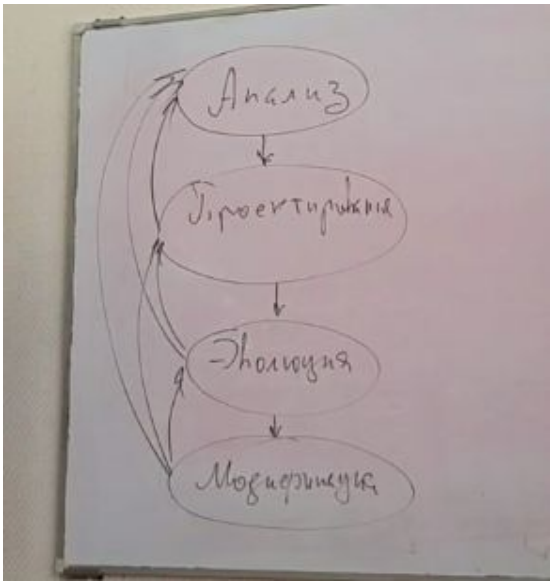
В ООП основной упор делается на легкость модификации программы.

Этапы:

1. Анализ (Чуть ли не основной этап) - построения модели нашей задачи (Проходит вместе с заказчиком). Требования к модели: Должна быть полная и понятна всем заинтересованным лицам.
2. Проектирование - перевод из документов, которые помогали нам проводить анализ в проектные документы на основе которых мы пишем код.
3. Эволюция - это кодирование и тестирование. Отличие от просто кодирования и тестирования в том, что мы используем рекурсивный дизайн, начинаем с простого, а потом ее развиваем. Если возникли проблемы переходим к проектированию (перероектируем) или к анализу. РД подход - рекурсивный дизайн. Эволюция это разработка программного продукта, а модификация это следующие изменения (Когда мы сдали заказчику, а он сказал что-то подправить)
4. Модификация (Этот этап как часть технологии) - этап после сдачи работы заказчику.

Эволюцию и модификацию выполняют разные люди.

В ООП используется рекурсивный дизайн. Мы из каждого этапа можем вернуться на предыдущий этап (рис ниже).



Плюсы эволюции: обширная обратная связь. Появляются разные версии нашей системы. Удобный механизм отката. Возможна реализация разных версия (альтернативных) (Для разных пользователей с разным функционалом). Появляется возможность плотно взаимодействовать с заказчиком (каждую версию можно обсудить с заказчиком). На начальных этапах есть результаты. Интерфейс проектируется отдельно (заранее, другими людьми). Можем перейти от одного решения к другому (быстро и легко).

Изменения при эволюции:

- Добавление нового класса (вещь безболезненная, которая не приводит к изменению написанного кода).
- Изменение реализации класса (можем реализовать на основе наследования. Добавляем новый класс, который подменяет другой класс. (изменяем реализацию не переписывая существующий код)).
- Изменение представления класса (можем с классом по другому работать) Не должно повлиять на переписывание кода.

След. этап реорганизация структуры - более болезненная часть, но затрагивает минимальное переписывание. Это использование каких-то паттерн (адаптер, компоновщик (Появляется), прокси можем использовать)

Если затрагивается изменения интерфейса класса (базового класса) это серьезная проблема - переделывание большого кода.

Какие документы создаются при объектно ориентированном анализе:

1. Документы, которые создаются для всей нашей программы
    - Схема доменов. Мы разбиваем задачу на домены.
    - Проектная матрица - документ, который в дальнейшем будем использовать при проектировании и эволюции. Нам нужна, чтобы мы четко оценивали на какой стадии разработки нашего ПО мы находимся.
2. Для каждого домена. Если домен большой, если кол-во классов в домене превышает 50 (лучше даже 30), то такой домен мы разбиваем на подсистемы по минимум связи.

   Мы разбиваем на подсистемы:
    - Модель связей подсистем.
    - Модель доступа к подсистемам.
    - Модель взаимодействия подсистем.
3. Для каждой подсистемы:
    - Информационная модель (Диаграмма сущность связь) - с этой модели мы начинаем проектировать.
    - Описание классов и их атрибутов (членов данных).
    - Описание связей.

   Выше три пункта относились к информационной модели.
    - Далее модель взаимодействия объектов (Или диаграмма).
    - Список событий, которые происходят в нашей подсистеме.

   Далее
    - Модель доступа к объектам.

    - Таблица процессов состояний.

4. Это было для подсистемы, а теперь документы, который создаются для каждого класса:
    - Модель состояний (диаграмма переходов состояний).
    - Алгоритмы действия состояний (когда мы формализуем состояния мы их получаем)

5. Дальше для каждого состояния:
    - Диаграмма потоков данных действий (При формировании этой диаграммы мы выделяем процессы (таблица процессов состояний(выше)).
6. Для каждого действия:
    - Описание процессов.

С чего начинать разработку:
Подход белого ящика в ооп.
Подход черного ящика в структурном.

Начинается все с информационного моделирования. Изначально разбили задачу на домены и рассматриваем прикладной домен (нашу задачу). Переходим к подсистемам, когда количество классов в прикладном домене становится критическим и когда четко выделяются группы объектов, которые мы можем разбить на подсистемы.
Но начинаем с информационного моделирования.

Начинаем с физических объектов, которые реально существуют в физической вселенной. Мы смотрим какие объекты существуют, пытаемся эти объекты сгруппировать по принципу одних и тех же характеристик. Есть объект, выделяем, что характеризует этот объект. Подходим именно не что нам делать, а пока рассматриваем чем характеризуются наши сущности. Выделяем атрибуты объектов - это характеристики.
Атрибуты:

- Идентифицирующие атрибуты. Идентификатор - набор из одного или нескольких атрибутов, которые четко идентифицируют объект (те атрибуты, которые мы объединили в понятие идентификатор явл. идентифицирующими атрибутами). (Их еще называют указывающие атрибуты)
- Атрибут может быть описательным (Вес, рост - характеристики студента, выступают как описательные).
- Вспомогательные атрибуты мы вводим при формализации связей (Это может быть атрибут состояние - статус)

Для каждого атрибута мы смотрим, какие значения он может принимать. В дальнейшем нам нужно понять, какого типа атрибут.

Каждый атрибут мы должны описать, должны понять, почемы мы его выделили и его описать. Описание атрибута - это текст, из которого становится понятным, зачем нам нужен данный атрибут - почему мы его выделили (данную характеристику).

Если это касается описательного атрибута, то мы показываем какую конкретную характеристику объекта сберегает этот атрибут. Также мы должны описать как определяется этот атрибут и кто задает значение этого атрибута.

Если это касается идентифицирующего атрибута - мы должны описать форму указания (если она уместна) кто назначает указания. И указать степень в которой этот идентифицирующий атрибут используется как идентификатор. Среди всех идентифицирующих атрибут мы выделяем привилегированный идентификатор (один атрибут).

Каждый объект должен быть уникальным. Его уникальность - это его идентификатор
(К примеру в группе студентов имя или фамилия не может быть, потому что в одной группе может быть несколько студентов с одинаковым именем или фамилией. Имя и фамилия - можно, потому что маленькая вероятность, что попадется в одной группе 2 студента с одной и той же фамилией и именем. Добавляем имя - формируем идентификатор. Привилегированный атрибут будет фамилия).

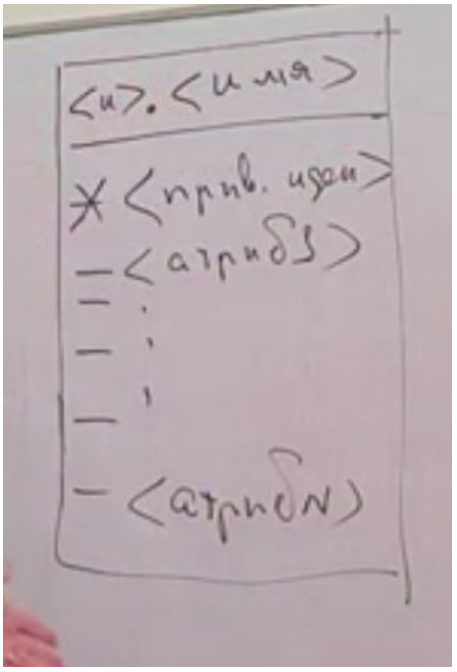
Изменения атрибута не приводит к изменения объекта(Вес изменился - объект остался тот же).

Если изменился идентифицирующий (указывающий), то просто другое имя стало и того же самого объекта (объект остался тот же самый)

Если изменяется вспомогательный объект, который формализует связь, то это говорит о том, что другие объекты участвуют в этой связи.

На диаграмме сущность связи (информационной моделе) мы доллжны сущности располагать в прямоугольниках. Каждой сущности присваиваться уникальный номер (должен быть уникальным в нашем домене). Далее мы выделяем имя сущности (К примеру студент, преподаватель). Далее идут атрибуты. Привилегированные атрибуты мы

помечаем звездочкой. Все остальные атрибуты мы просто перечисляем (рис. ниже (информационная модель сущность-связь)).
(Еще нужен ключевой литерал - будем им помечать события, которые происходят, чтобы не расписывать полностью имя сущности)



Для каждого атрибута мы выделяем правила атрибутов - формируем список значений которые может принимать данный атрибут.

Первое правило: Для данного объекта каждый атрибут должен иметь значение в любой момент времени (значение должно быть единственным). Если какой-то атрибут мб не определен, то это говорит о том, что это другая сущность.

Второе правило: При информационном моделировании не один атрибут не должен содержать внутренней структуры (он не должен быть сложным (вес, рост)).

Третье правило: Когда атрибут имеет составной идентификатор, каждый атрибут, который явл. частью идентификатора представляет характеристику всего объекта, а не его части (вот студент, ФИО - это все характеристики всего объекта, не его части, и ничего другого).

Четвертое правило: Если атрибут не явл. частью идентификатора, то он должен представлять характеристику данного объекта, указанного идентификатором, а не характеристику какого-либо другого атрибута.

Между сущностями возможны связи. Мы формально подходим к этим связям. Мы не говорим что конкретно, но мы говорим, что существует связь (студент - преподаватель)

Пример:

1. Преподаватель (П - ключевой литерал).

Идентификатор - * имя.

Характеристики - стаж, ученая степень, звание и тд….

2. Студент (С - ключевой литерал)

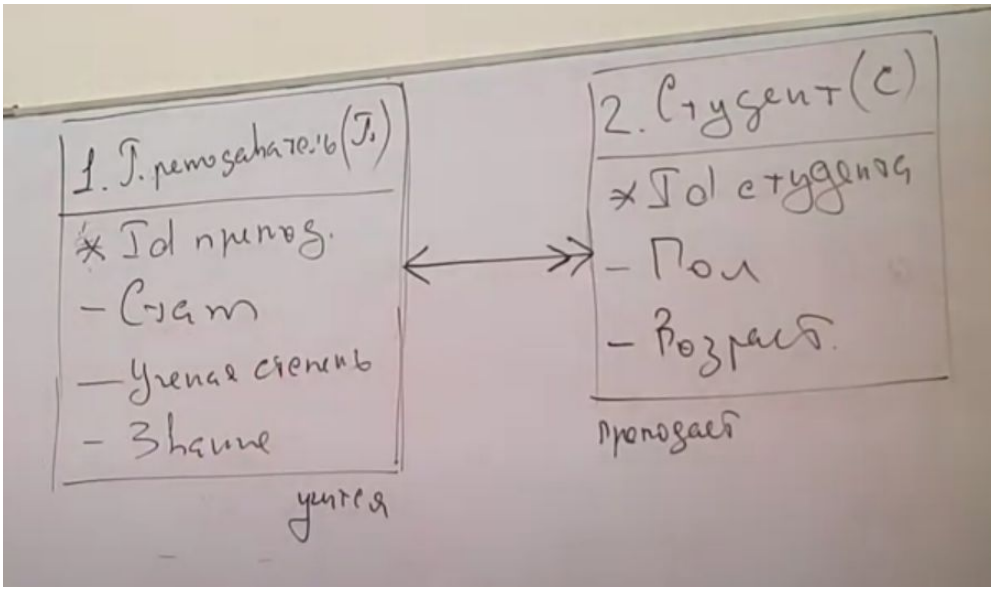
Идентификатор - * имя.

Характеристики (атрибуты) - пол, возраст, и тд…

Между этими сущностями возникает связь. И мы должны задать задать эту связь из перспективы каждого участвующего объекта.
Преподаватель что делает? Преподает студенту.
Студент? Учится у студента.

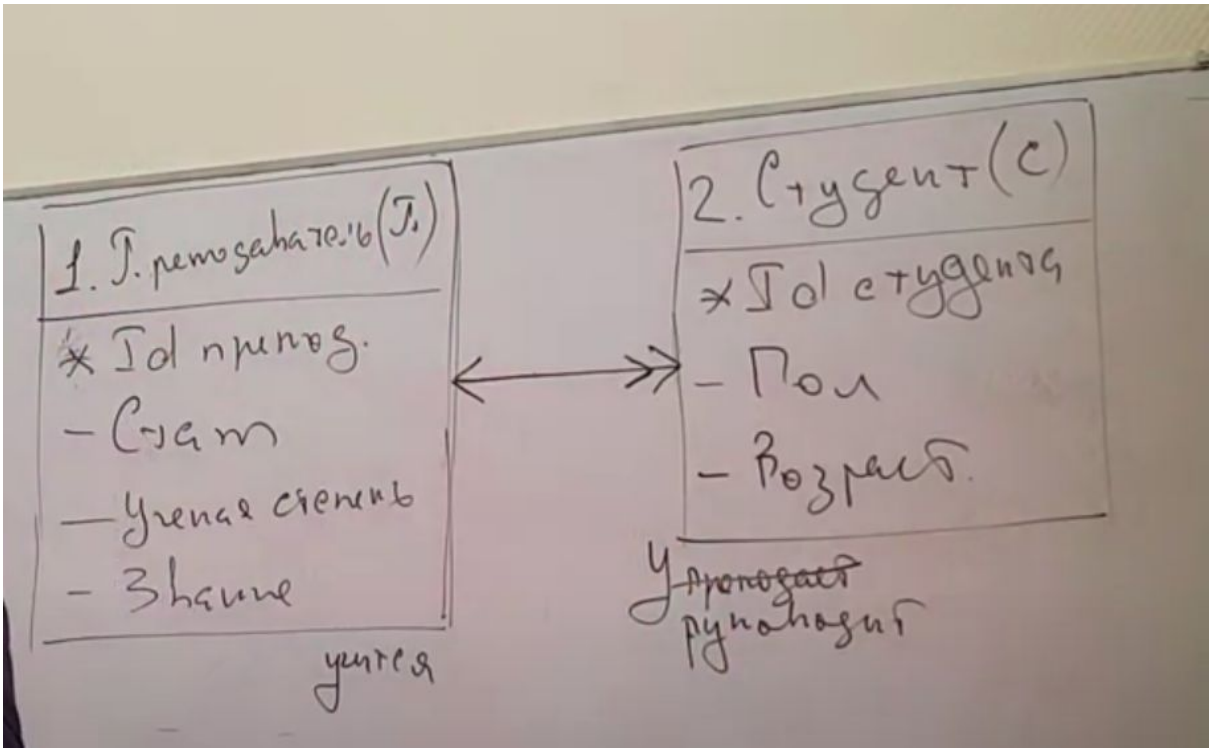
Данная связь один ко многим (Один руководитель и много студентов (Руководит несколькими студентами один преподаватель). - тогда просто линяя.
Если связь один к одному (муж и жена), то на диаграмме рисуем стрелки на концах:

Если один ко многим, мы показываем, что это много (две стрелочки)



Возможна связь многие ко многим (пример: учебный курс и студент, один учебный курс могу прослушивать много студентов и один студент может прослушивать несколько учебных курсов)

Иногда могут объекты не учавствовать в связи. Ставим букву “У” (**условная связь**) со стороны не участвующего объекта в связи. (Руководитель - студенты)

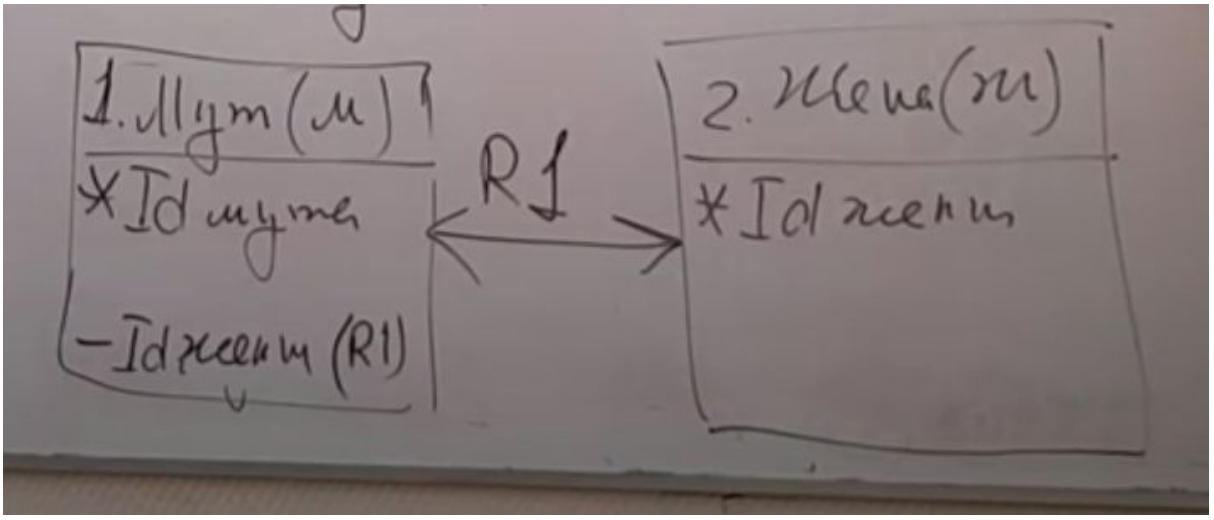


Биусловная связь - с обоих сторон не участвуют в связи (редкая ситуация).

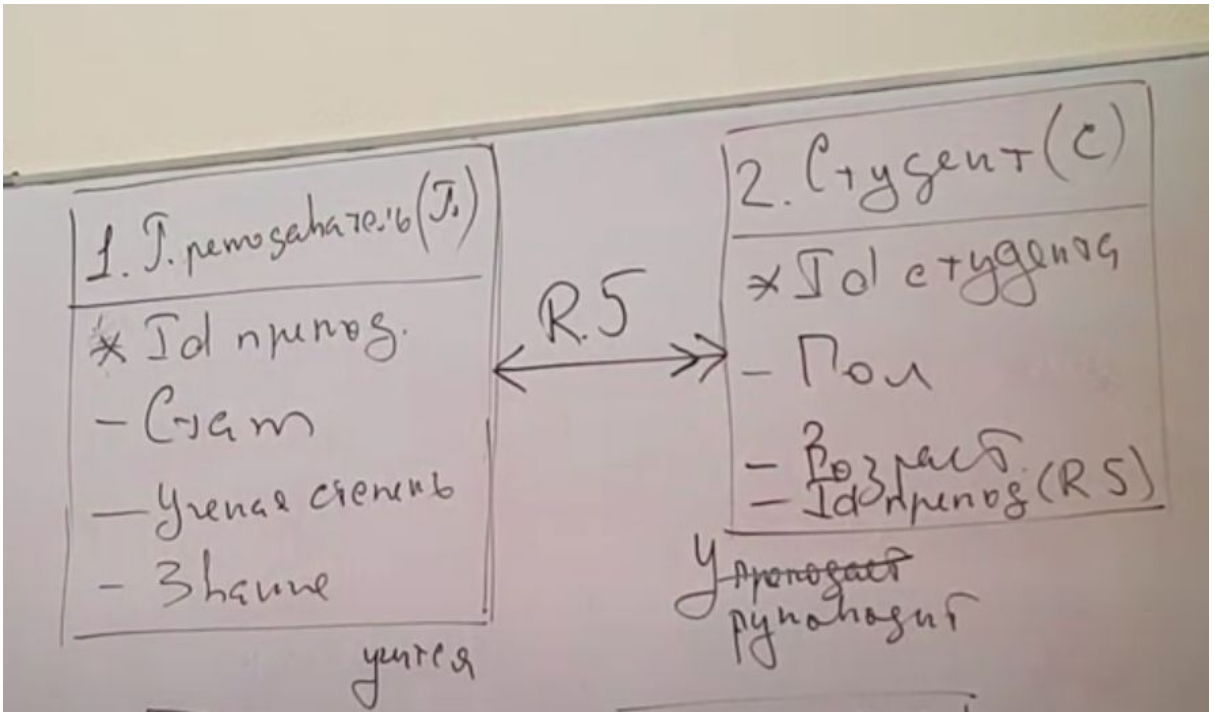
Связь должны формализовать. Кто-то должен отвечать за эту связь

К примеру связь муж-жена: Мы добавляем вспомогательный атрибут к одному из участвующий объектов. К тому, который наиболее осведомлен. Кто главный? Жена или муж (Не все согласны….) Пусть будет муж. Мы добавляем id жены к мужу.. Мы должны указать какая связь и для этого мы помечаем каждую связь идентификатором, (если внутреняя начинается с R), а дальше идет номер (уникальный в нашем домене). И мы помечаем, что данный атрибут формализует эту связь (R1 (рис. ниже)) устанавливает эту связь.

Это безусловная связь 1 к 1. Т.к. связь безусловная то на 1 из объектов накладывается условие жизни другого объекта.



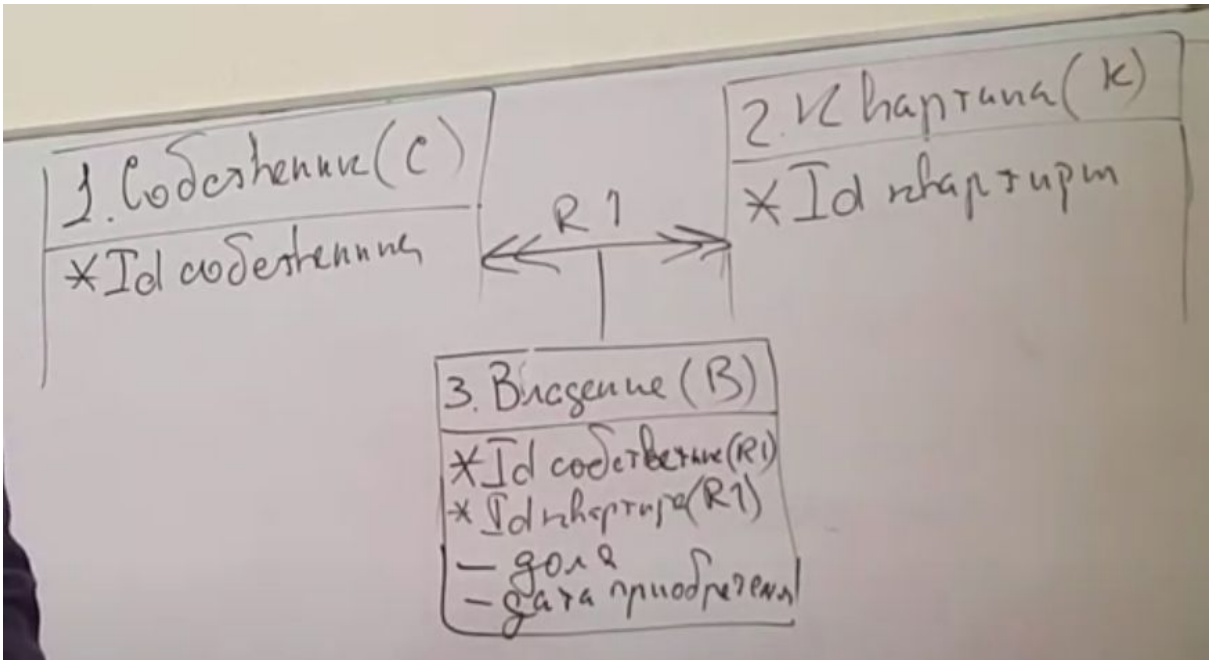
Если связь один ко многим. Мы должны ее тоже формализовать со стороны студента т.к. правило 2 гласит, что атрибут должен быть простым.



Как быть со связью многие ко многим? Для этого добавляют ассоциативный объект, задача которого формализовать эту связь.

Пример:

Допустим у каждого есть квартира. У кого-то может быть несколько собственников. Также у одной квартиры может быть несколько собственников. Сущность многие ко многим.
Собственник - квартира
Мы создаем ассоциативный объект - владение. Идентификатор владения явл. идентификатор собственника так и идентификатор квартиры. Ассоциативный объект, кроме атрибутов, формализующих связь (многие ко многим) может иметь дополнительные атрибуты. К примеру доля владения данной квартирой, дата приобретения.



Теперь к мужу и жене. Умирает муж (он держит связь). Как быть с женой? Объекта жены без мужа нет. Поэтому после смерти муж переводит жену в класс вдова. А если жена умирает, то муж без жены (за связь отвечает муж)? Если мы знаем, что время жизни совпадает полностью двух объектов, тогда мы используем связь 1 к 1 с помощью добавления вспомогательного объекта к одному из объектов (к главному).

Но если время жизни объектов не совпадает и нам важно что объекты могут появляться и исчезать, то нам нужен объект, который будет отвечать за эту связь (посредник) - ассоциативный объект. Он будет держать эту связи (отвечать за эту связь). Тогда не возникнет проблем с жизненными циклами.

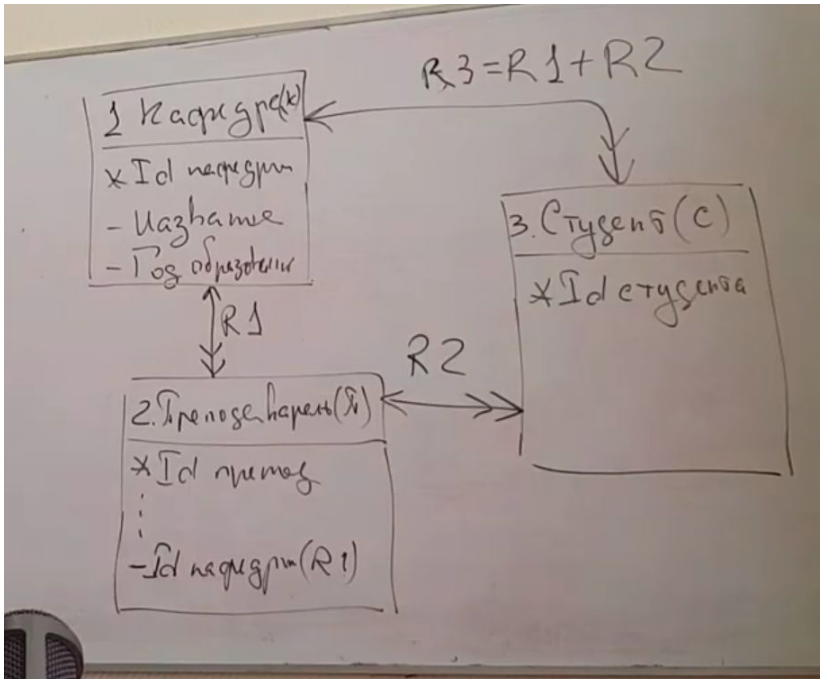
Проблема при связи один ко многим: Если со стороны многих находится объект, который не может отвечать за всю систему. В этом случае мы можем тоже формализовывать связь ассоциативным объектом. (лекция 10 часть 3 мин 22.45).

Вывод: мы ассоциативным объектом формализуем связи многие ко многим и любую связь, которая имеет динамическое поведение (может меняться во время работы программы). За это отвечает объект посредник. Когда мы четко должны выстроить иерархию кто главный а кто нет (Это главенствование передаем посреднику). При работе со связями мы работаем через объект владение (ассоциативный объект) (выше пример).

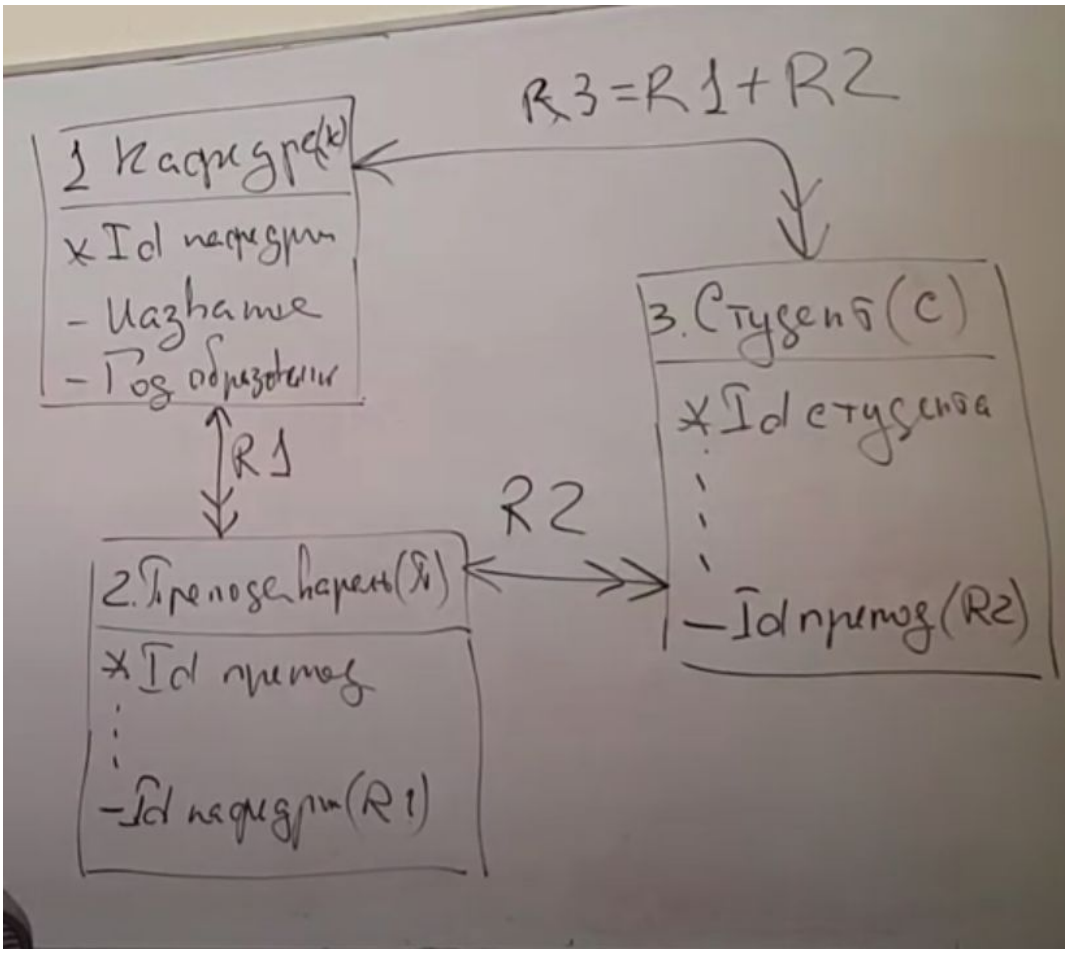
Что нам для связей нужно?

1. Идентификатор связи.
2. Формулировка связей со стороны каждого участвующего объекта.
3. Вид связи (1 к 1 / 1 ко многим / многие ко многим).
4. Формализация связи (Есть вспомогательный атрибут или ассоциативный объект)
5. Формулировка основания связи (Зачем нам нужна эта связь?)

Некоторые связи могут быть следствием других связей. (R3 - композиция других связей)



Когда приходим к такому “треугольнику” мы одну из связей формализуем как композицию других. Избыточности в связи не должно быть. Те связи, которые явл. следствием других связей мы на информационной модели (на диаграмме сущность связь) не формализуем. Т.е. формализуем только ту связь, которая **не** явл. композицией: (Добавили id преподавателя (не кафедры)).



Подклассы и супер классы.

Атрибуты, которые явл. общими для разных классов выносит в супер класс. А все остальные атрибуты, которыми отличаются объекты, они будут в подклассах.

В ООП супер класс всегда абстрактное понятие. Мы не рассматриваем объекты супер класса (абстрактного класса).

Пример: Есть разные баки (с подогревом, для смешивания продуктов,...)

1. Суперкласс: Бак (Б) - абстрактное понятие.
Идентификатор:
* id бака.
Атрибуты:
- ёмкость.
- Дата изготовления.

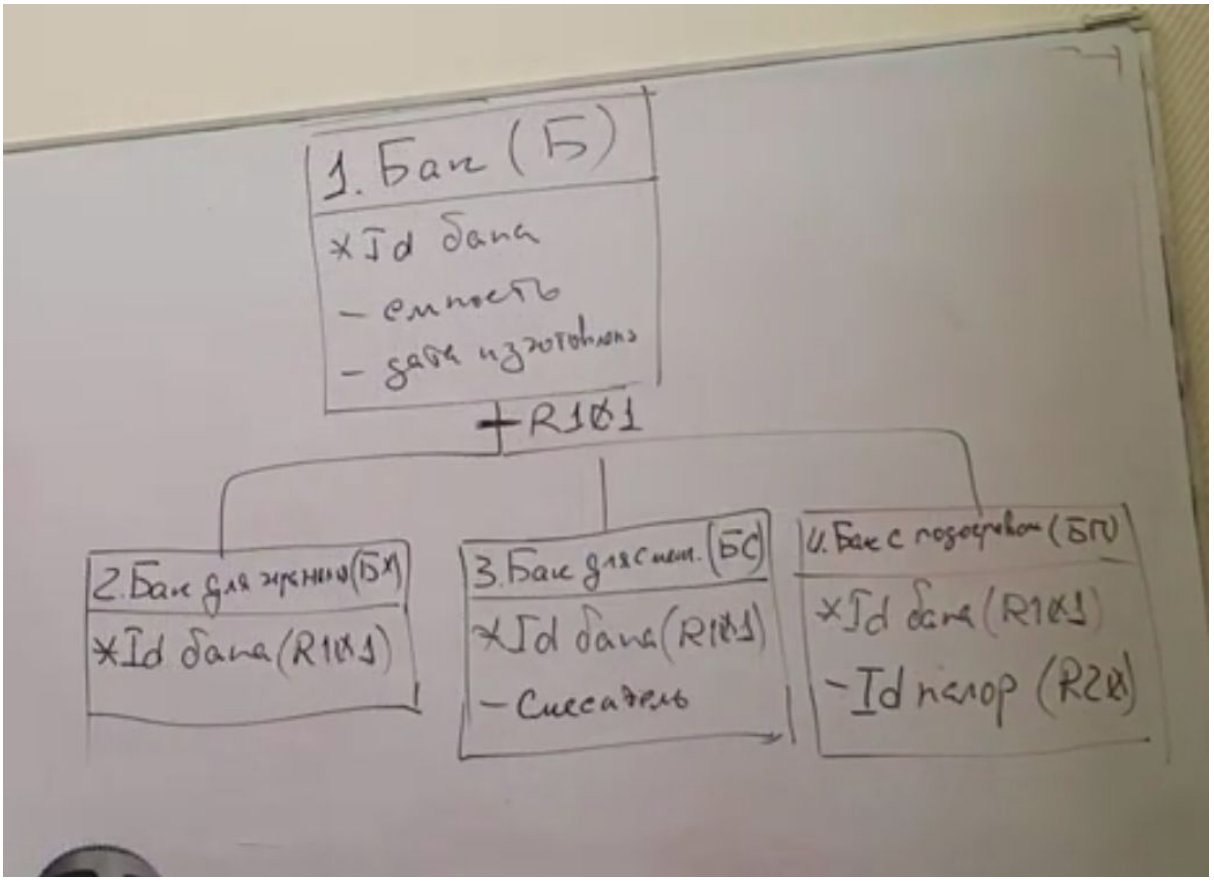
Подклассы:
2. Бак для хранения (БХ)
* id бака.

3. Бак для смешивания (БС)
* id бака.
- Смеситель.

4. Бак с подогревом (БП)
* id бака.
- Калорифер.

! Общие атрибуты выносится на уровень супер класса.

- На диаграмме рисуется связь и она отмечается черточкой и ставится идентификатор (Как правило для подкласса супер класса связь начинается с цифры 100) (Рис. ниже информационной модели)



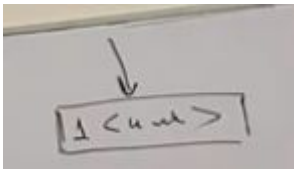
Это все выполняется на уровне информационного моделирования.

Далее смотрим на поведения объектов (не для всех можем выделить поведение). Отталкиваемся от реального физического мира. Характерная черта объекта - то как он проходит через стадии (ребенок, юношество, взрослый…). Объект переходит из одной стадии в другую скачкообразно (Пример: два состояния - сон и бодрствования. Возникает стадия просыпаемся - это переходная стадия.) Не все переходы возможны.

В физическом мире сущ. инциденты, которые заставляют объект переходить из одной стадии в другую или эти инциденты являются результатом переходом объекта из одной стадии в другую. Инцидент приводит к измененияю состоянию или явл. результатом переходом объекта из одной стадии в другую.

Модель состояний можем реализовывать или диаграммой или таблицей. (лекция 10 часть 4 где-то 22 минута)
ДПС - диаграмма переходов состояний.

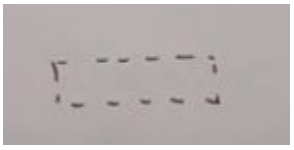
Каждое состояния рисуется прямоугольником. Каждому состояния присваивается уникальный номер и имя состояния.



Виды состояний:

1. Состояния создания. В этом состоянии объект появляется первый раз. Переход в такие состояния происходит не из состояния.
2. Заключительное состояние. Два варианта:

   1. Объект уничтожается. Это состояние рисуется пунктирной линией:

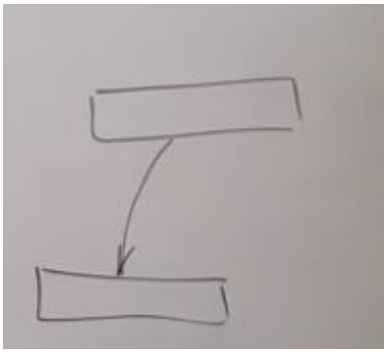


   2. Возможно состояние из которого объект в другие состояния не идет, но он не уничтожается. Рисуется таким же прямоугольником, только переходов в другие состояния из этого нет.

3. Текущее состояние, которое не относится ни к состоянию созданию ни к заключительному состоянию. Чтобы определять текущее состояние мы в класс добавляем вспомогательный атрибут, который определяет это состояние. Его называют статусом.

Переход из одного состояния в другой происходит в результате возникновения какого-то события (инцидента)

Событие рисуется стрелкой. Из того состояние, которое было в которое мы переходим.



1. Значение события.
2. Предназначение события.
3. Метка события, номер.
4. Данные события.

С каждым состоянием мы связываем действие, которое должно выполняться при переходе объекта в состояние.

Правила связи события с состоянием:

1. Все события, которые переводят объект в определенное состояние должны нести одни и те же данные.
2. Правило состояния создания. То событие, которое переводит в состояние создания не несет идентификатора объекта.
3. Правило состояния не создания. Оно должно нести обязательно идентификатор объекта как данное.

Каждому состояние мы ставим в соответствие действие. Задача действия перевести объект в то состояние, которому оно соответствует.

Действие может выполнять любые операции над атрибутами самого объекта. Но кроме того выполнять любые вычисления. Может порождать события для любого объекта любого класса (в том числе и для самого себя). Выполнять любые действия над таймером (создавать, удалять, запускать таймер, очищать). На данном этапе нет ограничений для действия (Действие имеет доступ к любым атрибутам объектов любых классов).

Действие должно:

1. гарантировать непротиворечивость самого объекта. После выполнения действия атрибута объекта не должны противоречить друг другу.
2. При создании и удалении объектов действие должно позаботиться о связях (они должны быть непротиворечивы)
3. Действие должно менять атрибут состояния статус в то, в которое мы переходим.

Для данного объекта в данный момент может выполняться только одно действие.

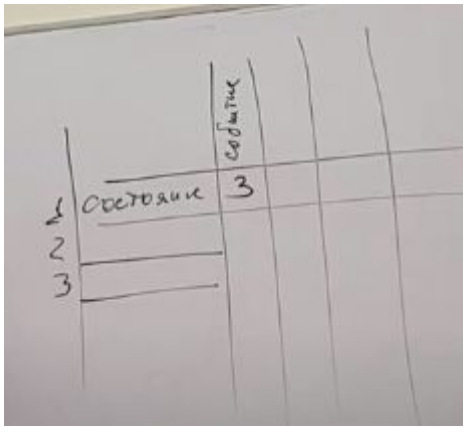
Таблица переходов в состояния (тпс)

С помощью этой таблицы мы контролируем, какие возможны переходы.

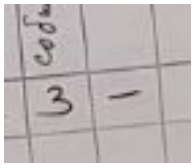
В тпс каждая строка это состояние в котором может находится объект. У каждого состояния есть уникальный номер.

Столбец это событие.

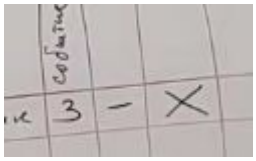
Пересечение состояния и события (строки и столбца) это состояние в которое объект перейдет в результате возникновения этого события в этом состоянии.



Если событие игнорируется (нет перехода) ставим прочерк.



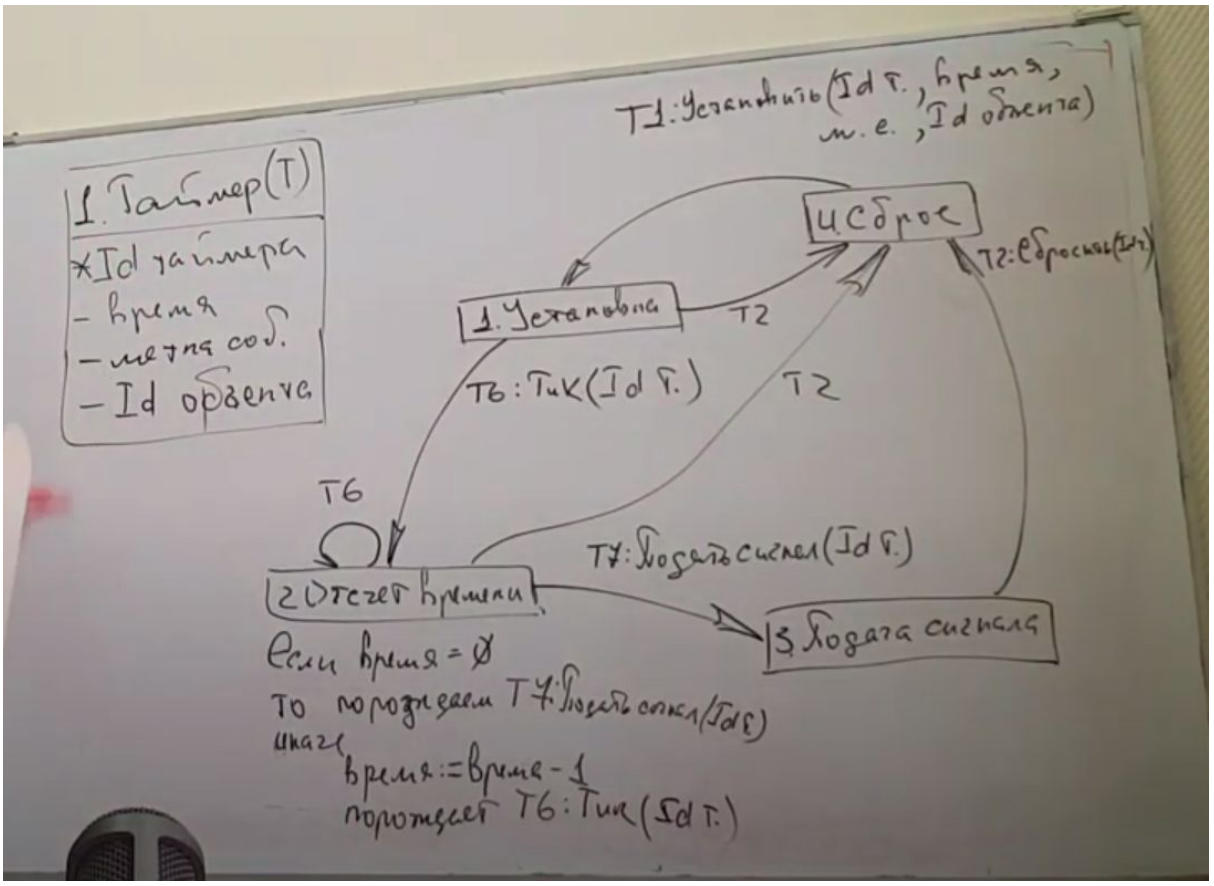
Если данное состояние не может произойти (нужно стремиться, чтобы их не было) то это крестик:

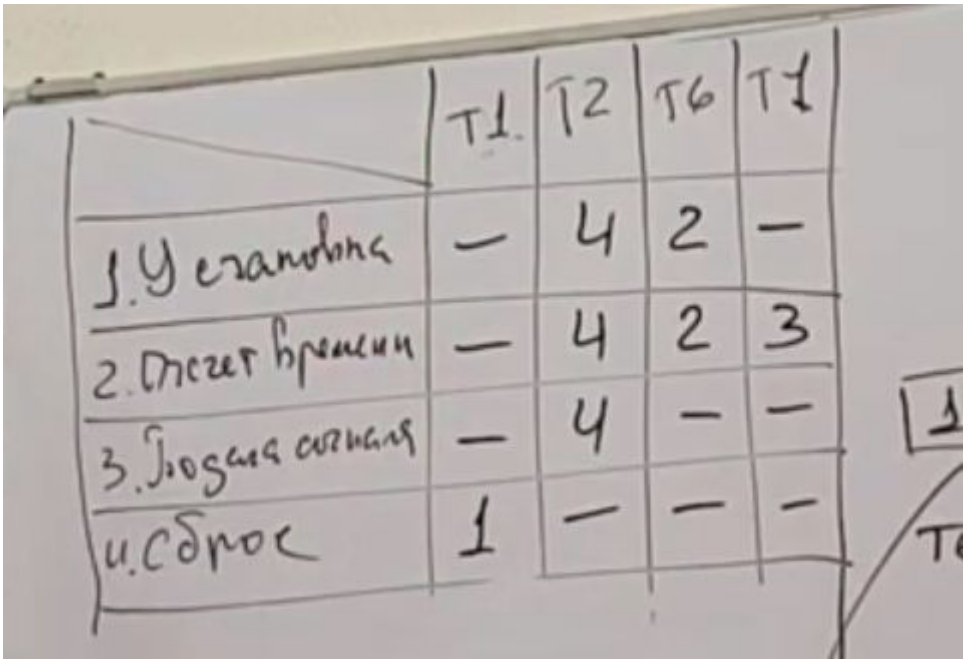


! Все ячейки должны быть заполнены !

Пример: Таймер.

T - идентификатор события.
1,2,... - номер события.





	T1	T2	T6	T7
1. Установка	—	4	2	—
2. Отсчет времени	—	4	2	3
3. Подача сигнала	—	4	—	—
4. Сброс	1	—	—	—

Могут возникать **состояния контекста** - это промежуточные состояния которые определяются предыдущим и следующим состоянием. Их цель для того чтобы организовать переход. Чтобы был скачкообразный переход. Двери закрыты, двери открыты, мы не можем перейти сразу в открытые двери должно быть состояние контекста (Закрываются). Переходное состояние определяет контекст (переходим из состояния сна в состояние бодрствование).

Если есть отличия в поведении, то относим к разным классам. Если жизненные циклы совпадают, то объединяем их суперклассом.

Когда выделяем жизненные циклы:

1. Если задача или запрос.
2. Если динамическая связь. Выделяем динамический объект и для него формируем жизненный цикл. выделяем состояния потому что связь динамическая.
3. Если объект создается поэтапно (Это строить (стройка, есть точные состояния стройки)).
4. Любое оборудование, которое имеет четко выделяемые состояния (лифт, самолет...)

Для некоторых пассивных объектов в интересах активных объектов мы тоже формируем жизненные циклы.

Далее анализ отказов. Мы должны проанализировать возможные переходы и постараться избавиться от крестиков. Все события должны порождаться и кто их порождает (нужно это проанализировать)?

Лифт ехал и застрял… Мы должны добавлять в жизненный цикл состояния соответствующие этим ситуациям, чтобы объект вывести из этой ситуации.

В результате получаем диаграмму и таблицу переходов состояния. Также получили описания действий (что будет происходить для каждого состояния). Также у нас есть список событий, которые должны происходить для нашей модели состояний.

Модель взаимодействия объектов (МВО).

Мы рассматриваем только нашу подсистему (домен).
Все события разделяем на 2 группы:

1. Те события, которые приходят извне.
2. Те события, которые происходят внутри нашей подсистемы.

На МВО мы ограничиваем нашу подсистему сверху и снизу.

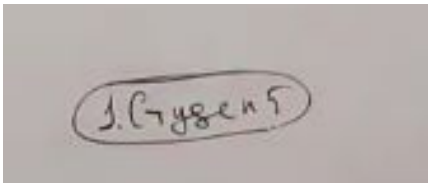
Наши модели состояний (объекты) рисуем в овалах.
Терминаторы - ограничения сверху и снизу.
Терминаторы могут быть верхнего и нижнего уровня.

Все модели состояний (объекты) мы строим так, что вверху располагаются объекты, которые более осведомлены о всей системе, а внизу менее осведомленные.

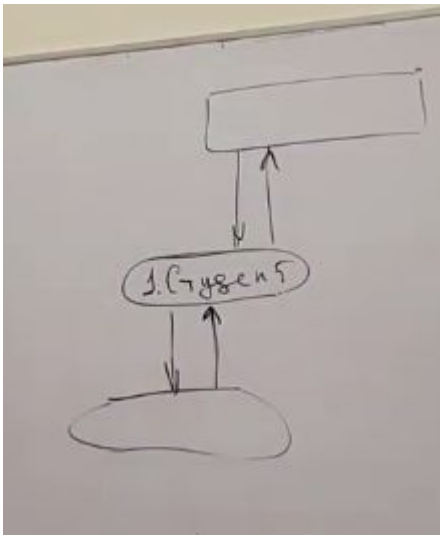
Внешние события, которые приходят от терминаторов, мы разделяем на 2 группы:

1. События незапрашиваемые, он не явл. следствием предыдущих действий нашей подсистемы.
2. Запрашиваемые, те события, которые явл. результатом действия нашей подсистемы

Модели состояний в овалах и записывается номер и какой сущности соответствует этот объект

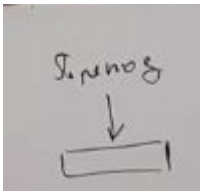


Приходит событие извне и объект наш может менять состояние. Мы должны проанализировать какие события принимает наша модель состояний и какие события она может порождать для других объектов.



В **прямоугольниках** мы рисуем состояние объекта.

Стрелка извне - это то что приходят незапрашиваемые события.



Важно модель построить с учетом временных рамок. Тогда мы выделяем 2 времени:

1. Время задержки - время, в котором объект должен находиться в этом состоянии. Записывается вне прямоугольника
2. И время выполнения действия, которое переводит объект в это состояние. (Двери закрываются). Внутри прямоугольника

Критично, когда время выполнения действия становится больше времени задержки.

Действие может породить событие для самого себя и для других объектов.

Важно чтобы подсистема пришла в окончательное состояние. Если наша подсистема приводит к тому, что возникает запрашиваемое событие, то мы его добавляем в канал управления (канал управления должен иметь конец (все объекты, которые мы выделили, должны прийти в окончательное состояние))

Процесс имитирования: мы должны сгенерировать все начальные состояния и для каждой генерации отработать все не запрашиваемые события (мы должны прийти в окончательное состояние).

4 л.р. моделирования мира состояний. Лифт.

Выделить состояния для лифта.
Выделить объекты, которые могут быть (кабин, двери, блок управления, пользователи лифтом и тд...)

Основной паттерн для реализации взаимодействия между объектами это паттерн подписчик издатель.
Два варианта реализации:

1. Асинхронное (основное взаимодействие).
2. Синхронное (для некоторых задач может быть).

Для лифта асинхронное взаимодействие, т.к. есть понятие время, время перемещения лифта между этажами, время открытия, время чтобы пользователь вошел в лифт или вышел.

Асинхронная схема взаимодействия - произошло какое-то событие, а объекты не сразу изменили с

вое состояние, а это происходит со временем.

К примеру лифт едет между этажами и пока что он не дойдет до нашего этажа он не сможет перейти в состояние открытия дверей (должно пройти время для перехода в состояние).