# Цели создания языка UML.

* Предоставить пользователям готовый к использованию язык визуального моделирования
* Предоставить механизмы расширения и специализации
* Быть независимым от определенного языка программирования и процесса разработки
* Интегрировать лучший практический опыт разработок

# Перечислить диаграммы языка UML.

* сценариев (use case diagram)
* классов (class diagram)
* состояния (statechart diagram)
* активности (activity diagram)
* последовательности (sequence diagram)
* коммуникации (collaboration diagram)
* компонентов (component diagram)
* топологии (deployment diagram)
* композитная структурная диаграмма
* обзорная диаграмма взаимодействия
* временная диаграмма
* диаграмма пакетов

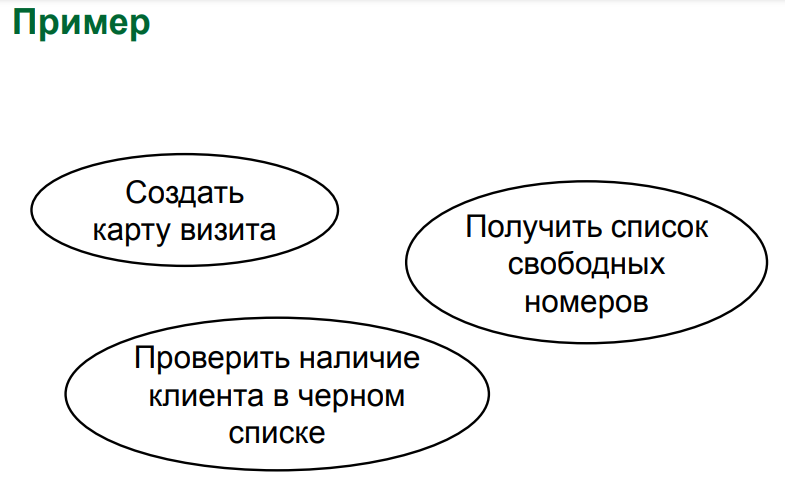
# Что описывает диаграмма сценариев?

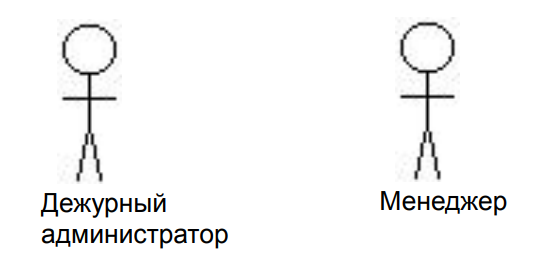
**Диаграммы сценариев описывают функциональное назначение системы** (то, что система будет делать в процессе своего функционирования)

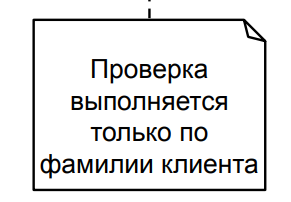
Диаграммы сценариев являются исходной концептуальной моделью системы в процессе ее проектирования и разработки

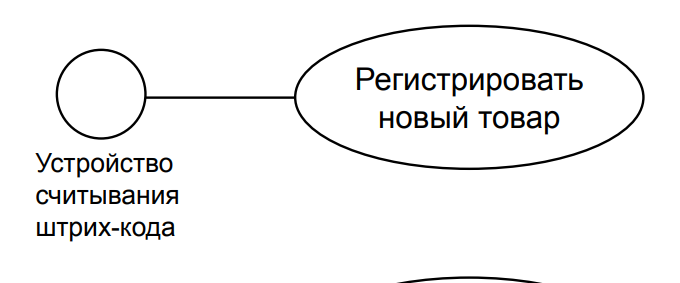
# Перечислить основные элементы диаграммы сценариев.

**Сценарий** – фрагмент поведения ИС без раскрытия его внутренней структуры



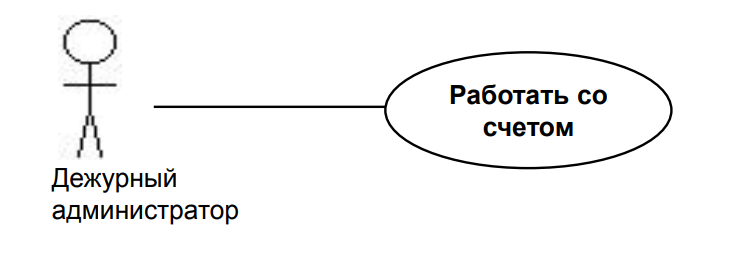
**Актер** представляет собой любую внешнюю по отношению к моделируемой ИС сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей

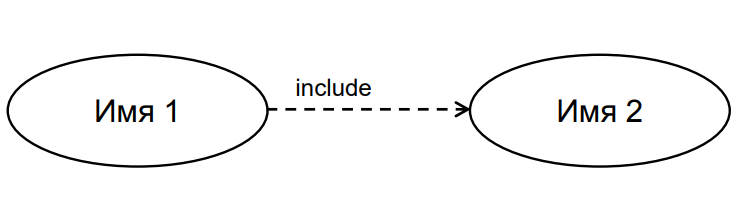
**Интерфейс** определяет совокупность операций, которые обеспечивают необходимый набор сервисов для актера

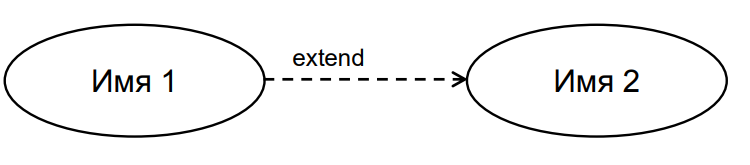


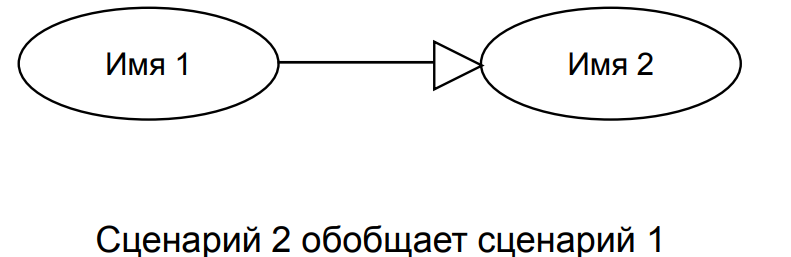
**Примечание** предназначено для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта

# Перечислить типы отношений на диаграмме сценариев.

* отношение ассоциации (association) 
* отношение включения (include)



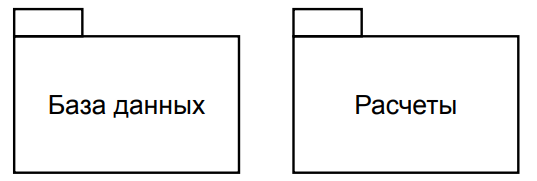
* отношение расширения (extend)  
  
* отношение обобщения (generalization)



# Что описывает диаграмма классов?

Диаграмма классов предназначена для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования

# Перечислить основные элементы диаграммы классов.

**Пакет** – способ организации элементов модели. Каждый элемент модели принадлежит только одному пакету.

**Класс** – обозначает множество объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов

**Свойство** <квантор видимости> <имя> [<кратность>] : <тип> = <исходное значение>

Кратность – кол во ([1..3,5,7..\*]). Квантор = [+-#].

* + color: RGB = (192, 192, 192)
* # navigable: boolean = TRUE
* + goal: enum(gTest, gWork) = gWork
  + id: integer
* + name [1..2]: string

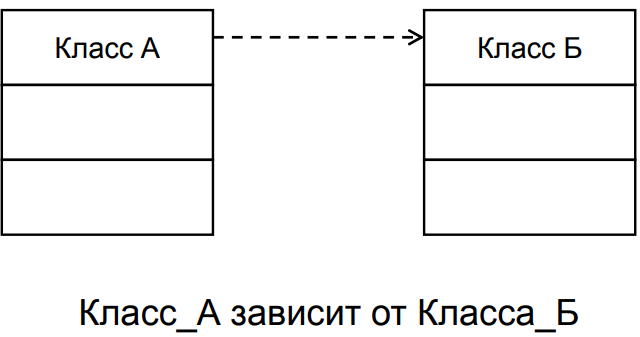
**Метод** <квантор видимости> <имя> (<список параметров>): <тип возвращаемого значения>

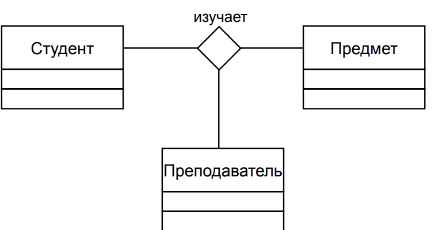
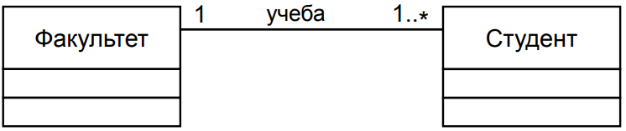
<**параметр**> <вид><имя> : <тип> = <значение по умолчанию>

* + создать()
* + нарисовать( in форма: Многоугольник =
* прямоугольник, in цвет\_заливки: Color =
* (0,0,255))
* – запросить\_счет\_клиента( in номер\_счета:
* integer): Currency

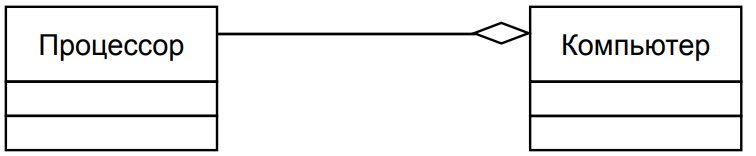
**Объект**

# Перечислить типы отношений на диаграмме классов.

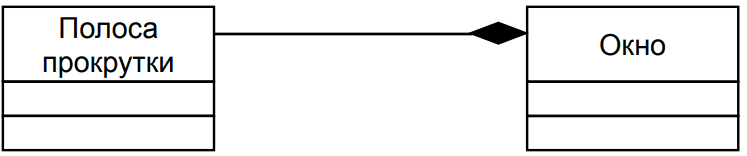
* отношение зависимости (dependency)   
  
* отношение ассоциации (association)

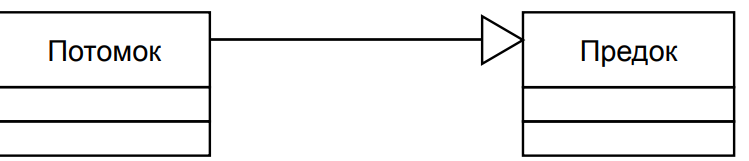


* отношение агрегации (aggregation)



* отношение композиции (composition)



* отношение обобщения (generalization)   
  
* отношение реализации (realization)

Между интерфейсами и классами

# Что описывает диаграмма состояний?

Диаграмма состояний описывает процесс изменения состояний только одного класса, а точнее – одного экземпляра класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта

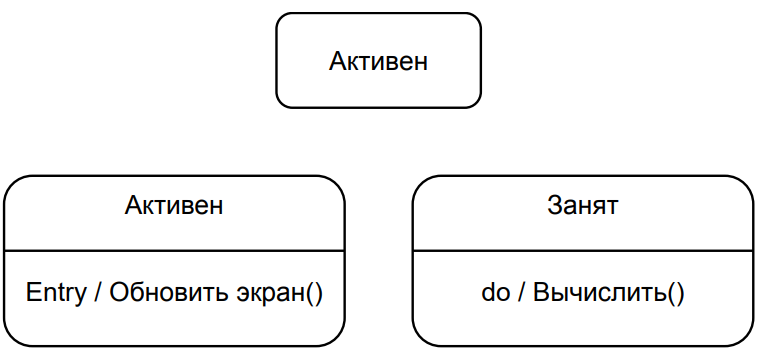
Диаграмма состояний – конечный автомат

# Что такое состояние?

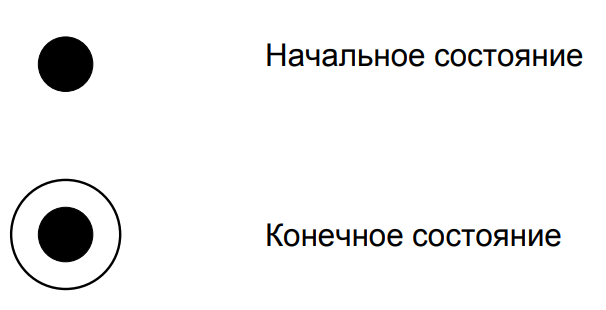
Состояние – набор конкретных значений атрибутов объекта

# Перечислить основные элементы диаграммы состояний.

Состояние



Действие = <метка> / <выражение действия>

<Метка>

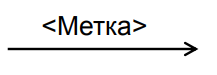
entry – вход в состояние

exit – выход из состояния

do – деятельность в состоянии

include – вызов подавтомата

**Переход** Переход осуществляется при наступлении некоторого события

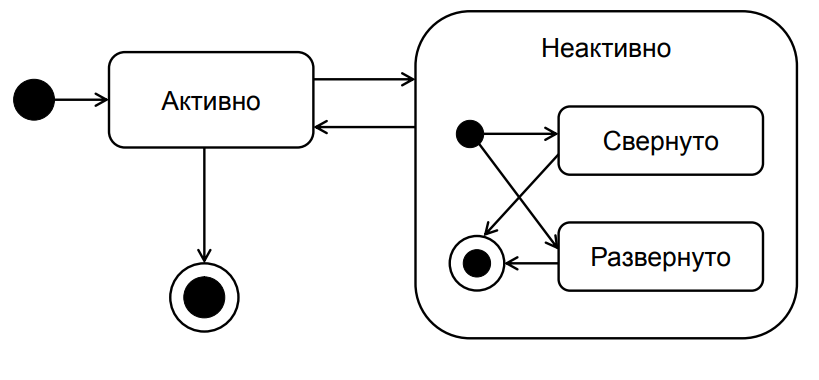


<**Метка**> ==== *<сигнатура события>[* <сторожевое условие> ]/ <выражение действия>

*<сигнатура события>* == <имя события> (<список параметров>)

[<сторожевое условие>]– булевское выражение

Переход:  

# Ограничения на диаграмме состояний.

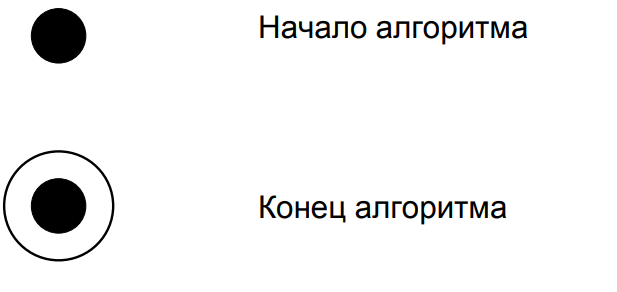
1. Переход из состояния в состояние происходит мгновенно
2. История переходов из состояния в состояние не запоминается
3. В каждый момент времени объект может находиться только в одном из своих состояний
4. В любом состоянии объект может находиться как угодно долго
5. Время на диаграмме состояний присутствует в неявном виде
6. Количество состояний должно быть обязательно конечным
7. Не должно быть изолированных состояний и переходов
8. Не должно быть конфликтующих переходов

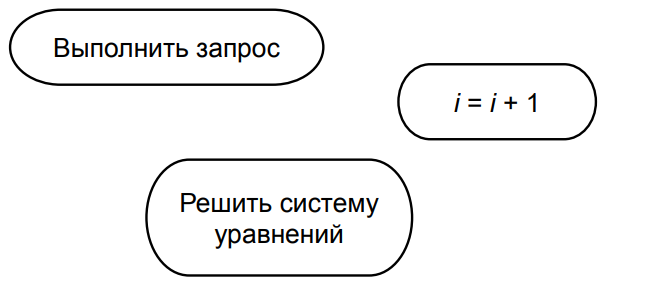
# Что описывает диаграмма деятельности?

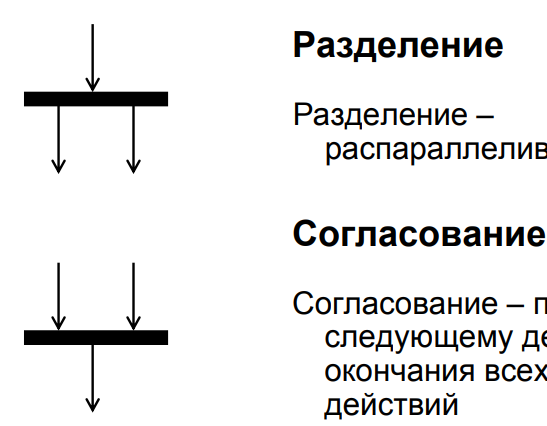
Диаграмма деятельности описывает процесс выполнения действий, т.е. логику или последовательность перехода от одного действия к другому

Диаграмма деятельности используется для моделирования бизнес-процессов

# Перечислить основные элементы диаграммы деятельности.

**Действие** – операция, выражение, вычисления и т.д.



**Переход** срабатывает сразу после завершения действия

**Ветвление** – разделение на альтернативные ветви.

**Соединение** – объединение альтернативных ветвей.

**Разделение** – распараллеливание действий

**Согласование** – переход к следующему действию после окончания всех согласуемых действий

# Что описывает диаграмма последовательности?

Диаграмма последовательности используется для представления временных особенностей передачи и приема сообщений между объектами

# Перечислить основные элементы диаграммы последовательности.

* Объект
* Линия жизни
* Фокус управления
* Сообщение
* Уничтожение объекта

# Перечислить типы сообщений на диаграмме последовательности.

* Вызов процедуры   
  Один объект вызывает процедуру и ожидает, пока она не закончится. Такое сообщение является синхронным. 
* Асинхронное сообщение

Объект передает сообщение и продолжает выполнять свою деятельность, не ожидая ответа. 

* Возврат из вызова процедуры  
  Объект передает сообщение об окончании выполнения процедуры.

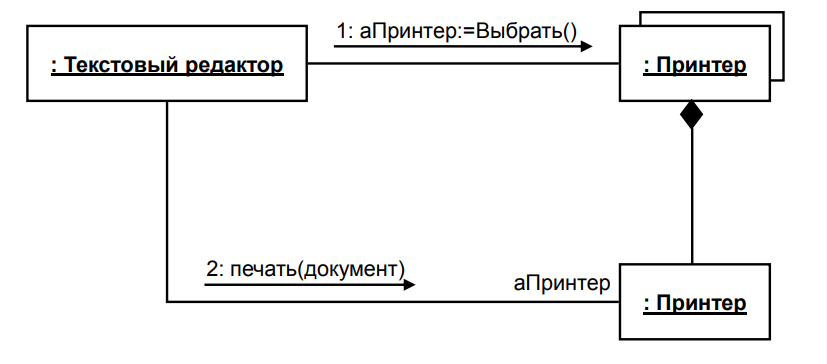


# Что описывает диаграмма коммуникации?

Диаграмма коммуникации (кооперации) предназначена для спецификации структурных аспектов взаимодействия объектов

# Перечислить основные элементы диаграммы коммуникации.

* Объект
* Ассоциация
* Сообщение



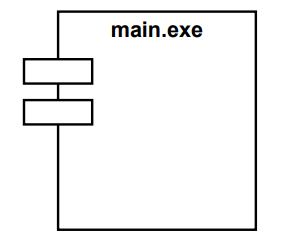
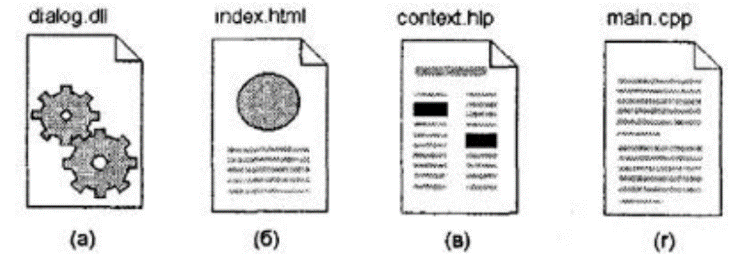
# Что описывает диаграмма компонентов?

Диаграмма компонентов описывает особенности физического представления системы

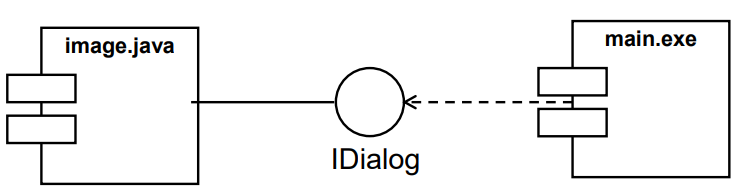
* визуализация общей структуры исходного кода программной системы
* спецификация исполнимого варианта программной системы
* обеспечение многократного использования отдельных фрагментов программного кода
* представление концептуальной и физической схем баз данных

# Перечислить основные элементы диаграммы компонентов.

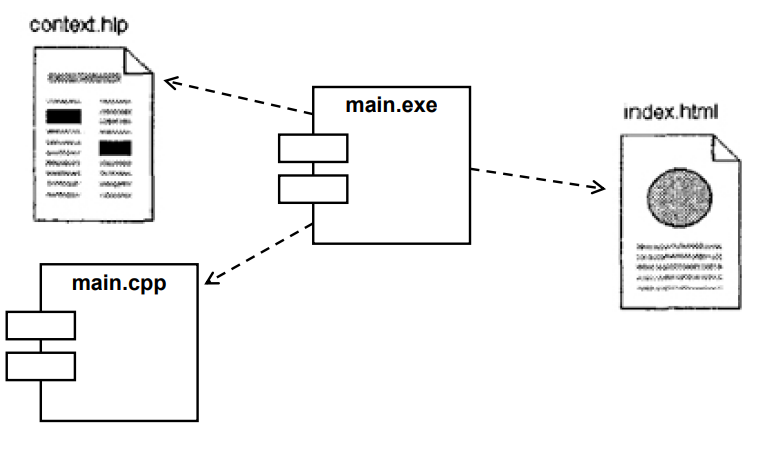
Компонент – крупно модульный объект: исполняемый файл, подсистема, документ и др



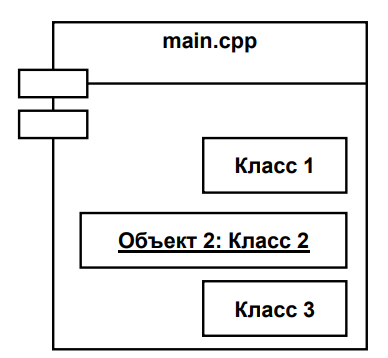
Интерфейс



Зависимость



# Как на диаграмме компонентов отображается реализация классов, интерфейсов?



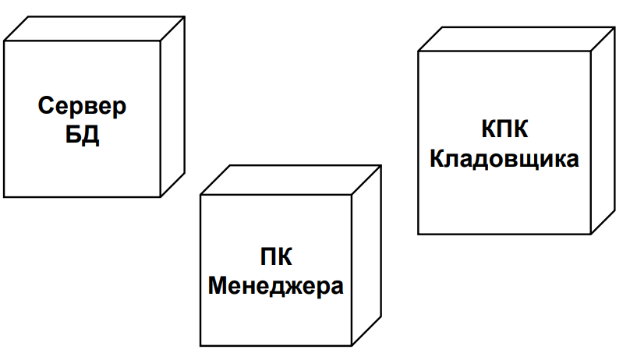
# Что описывает диаграмма топологии?

Диаграмма топологии применяется для представления общей конфигурации и топологии распределенной программной системы и содержит распределение компонентов по отдельным узлам системы

* определить распределение компонентов системы по ее физическим узлам
* показать физические связи между всеми узлами реализации системы на этапе ее исполнения
* выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности

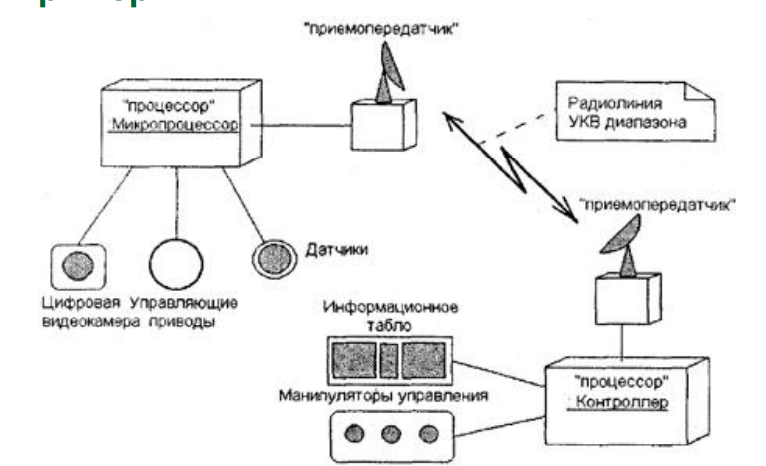
# Перечислить основные элементы диаграммы топологии.

Узел – физически существующий элемент системы:

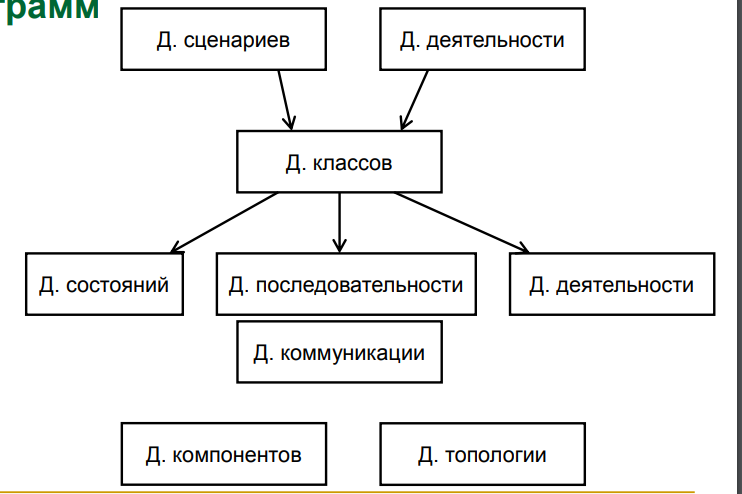
* Сервер
* рабочая станция
* принтер
* цифровая камера
* и др

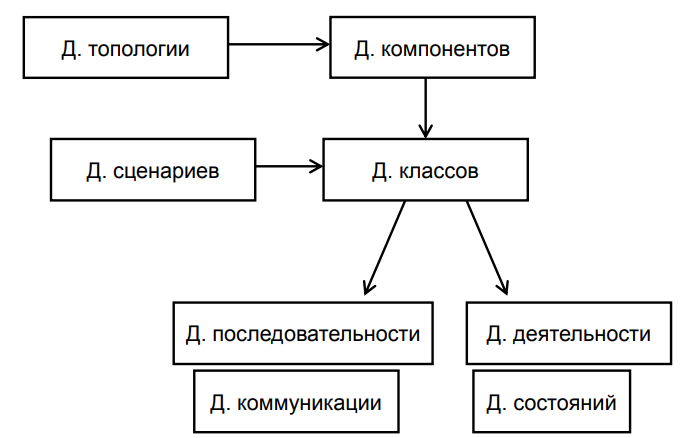
ассоциации

компоненты



# В какой последовательности разрабатываются диаграммы UML?





# Что такое CASE системы?

CASE (Computer Aided Software Engineering) – программные средства, поддерживающие процессы создания и сопровождения ИС

# Какими функциями обладают CASE системы?

В функции CASE входят средства анализа, проектирования и программирования программных средств, проектирования интерфейсов, документирования и производства структурированного кода на каком-либо языке программирования.

CASE-инструменты классифицируются по типам и категориям.

Классификация по типам отражает функциональную ориентацию средств на те или иные процессы жизненного цикла разработки программного обеспечения, и, в основном, совпадают с компонентным составом крупных интегрированных CASE-систем, и включает следующие типы:

* средства анализа — предназначены для построения и анализа модели предметной области;
* средства проектирования баз данных;
* средства разработки приложений;
* средства реинжиниринга процессов;
* средства планирования и управления проектом;
* средства тестирования;
* средства документирования.

Типичными CASE-инструментами являются:

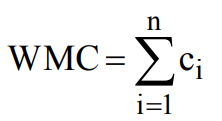
* инструменты управления конфигурацией;
* инструменты моделирования данных;
* инструменты анализа и проектирования;
* инструменты преобразования моделей;
* инструменты редактирования программного кода;
* инструменты [рефакторинга](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3) кода;
* генераторы кода;
* инструменты для построения UML-диаграмм.

# Перечислить характеристики объектно-ориентированных систем.

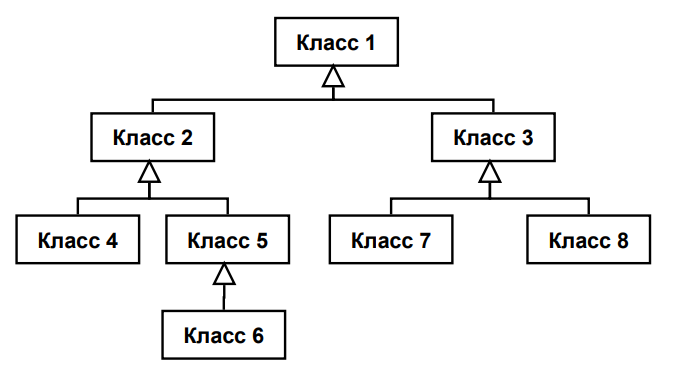
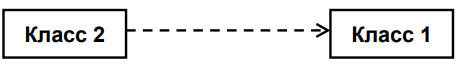
* Локализация
* Инкапсуляция – сокрытие данных. класс должен состоять из двух частей: интерфейса и реализации.
* информационная закрытость  
  Информационная закрытость делает невидимыми операционные детали программного компонента. Другим компонентам доступна только необходимая инфорымация.
* наследование — это механизм системы, который позволяет наследовать одними классами свойства и поведение других классов для дальнейшего расширения или модификации.
* абстракция это механизм, который позволяет проектировщику выделять главное в программном компоненте (как свойства, так и операции) без учета второстепенных деталей.

# Описать метрики Чидамбера и Кемерера.

<https://studfile.net/preview/1544366/page:3>

* взвешенные методы на класс  
  

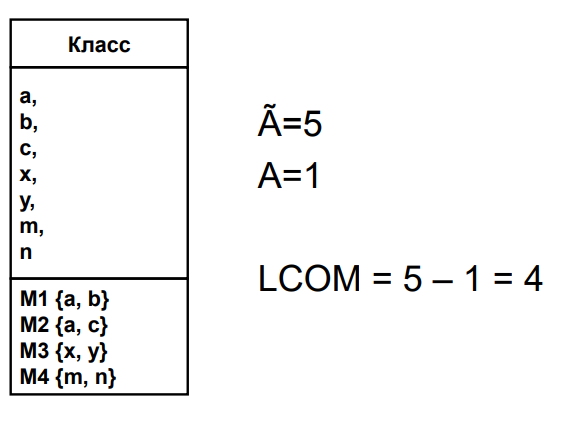
n методов со сложностями ci.

* высота дерева наследования  
  Макс длина от листа до корня. (=3)  
  
* количество детей (ветвления класса)
* сцепление между классами объектов  
  

Количество таких соединений (использований одного класса внутри другого)

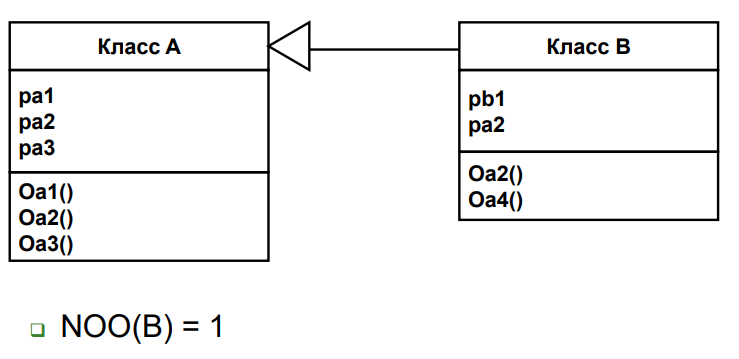
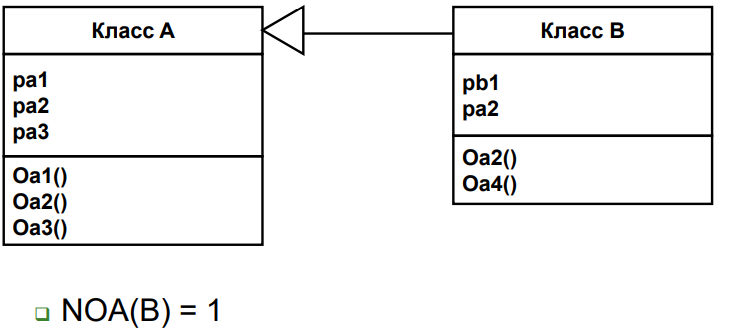
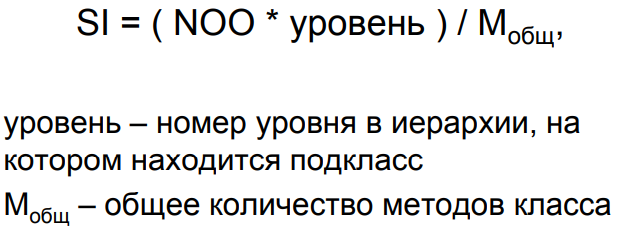
* отклик для класса  
  RFC — это количество методов класса плюс количество методов других классов, вызываемых из данного класса.
* недостаток связности в методах

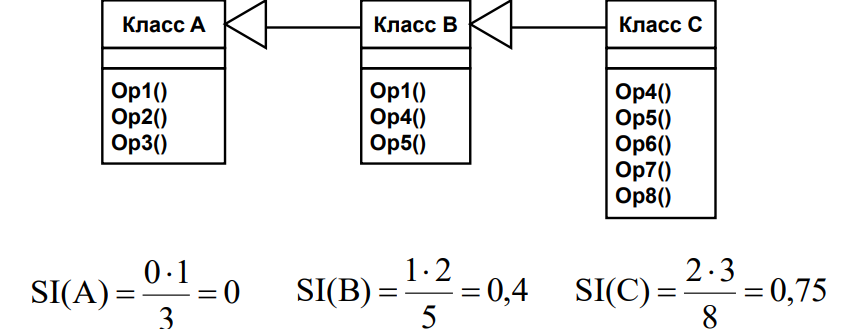
Насколько методы не связаны друг с другом



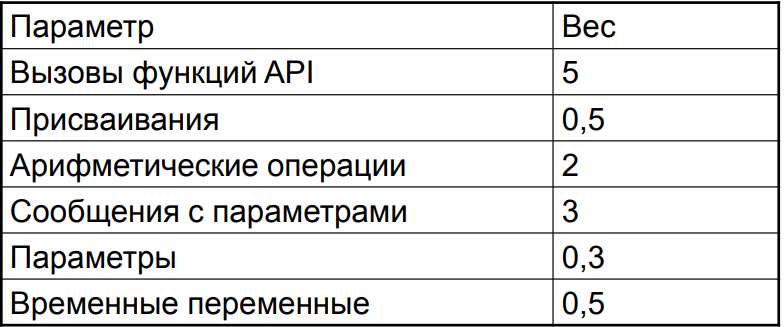
A – пары методов

# Описать метрики Лоренца и Кидда.

1. размер класса  
   Кол-во инкапсулированных операций + свойств
2. количество операций, переопределенных подклассом (NOO)  
     
   
3. количество операций, добавленных подклассом (NOA)  
   
4. индекс специализации  
   



1. средний размер операции  
   Кол-во строк прог.
2. сложность операции



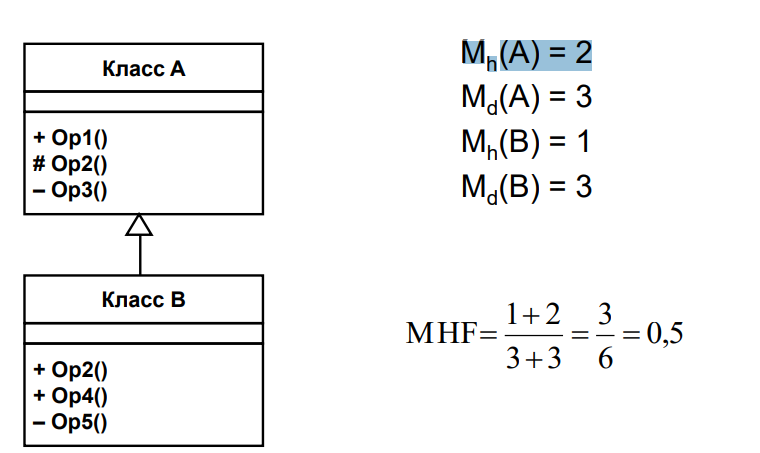
1. среднее количество параметров на операцию
2. количество описаний сценариев  
   количество классов, необходимых для реализации требований

* Это количество прямо пропорционально количеству классов, требуемых для реализации требований, количеству состояний для каждого класса, а также количеству методов, свойств и сотрудничеств. Метрика NSS — эффективный индикатор размера программы.
* Рекомендуемое значение NSS — не менее одного сценария на публичный протокол подсистемы, отражающий основные функциональные требования к подсистеме.

1. количество ключевых классов
2. количество подсистем

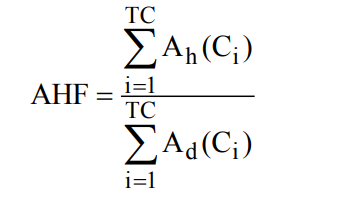
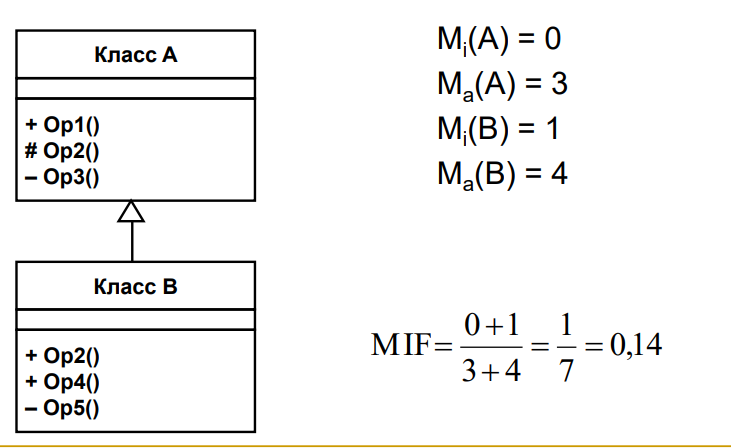
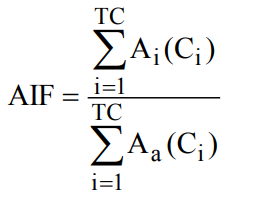
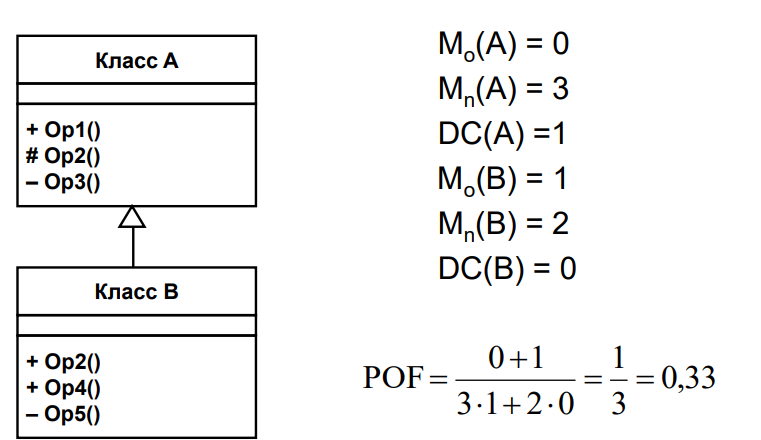
# Описать метрики Абреу.

* фактор закрытости метода



Mh – скрытые методы

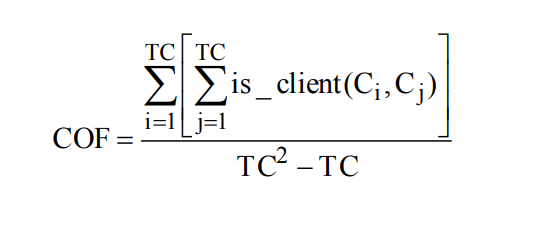
Md – Всего методов

* фактор закрытости свойства  
  
* фактор наследования метода  
  
* фактор наследования свойства  
  
* фактор полиморфизма  
  

Mn – количество новых

M0 – количество унаследованных и переопределенных

DC – количество потомков

* фактор сцепления  
  

# Для чего и как применяются метрики объектно-ориентрованных систем?

Для оценки качества разработанных моделей используется специальный метрический аппарат

Для объектно-ориентированных систем выделяют пять характеристик качества: локализация, инкапсуляция, информационная закрытость, наследование, абстракция

Существует большое количество метрических систем