

# UML - унифицированный язык моделирования

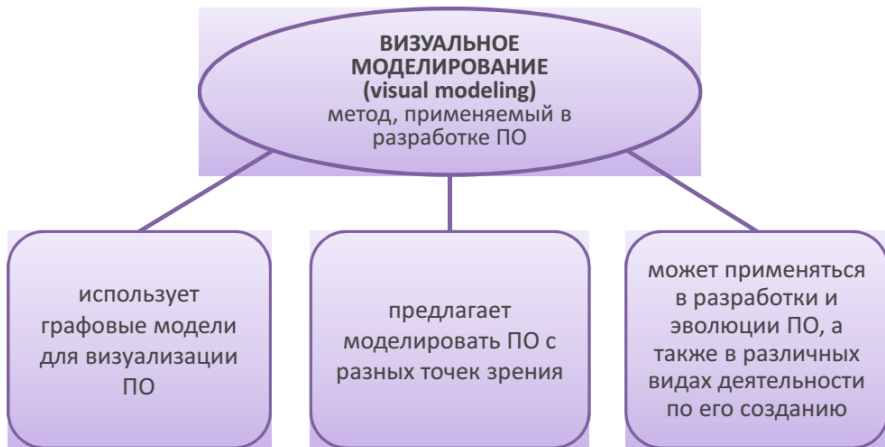
Наумов Д.А., доц. каф. КТ

Основы программной инженерии, 2019

# Содержание лекции

- 1 Визуальное моделирование и его средства
- 2 Понятие и применение UML
- 3 Виды диаграмм UML
  - Диаграмма вариантов использования
  - Диаграмма классов
  - Прочие диаграммы, пример

# Визуальное моделирование и его средства



# Язык UML

## Унифицированный язык моделирования (UML)

язык графического описания программных сущностей в виде диаграмм.

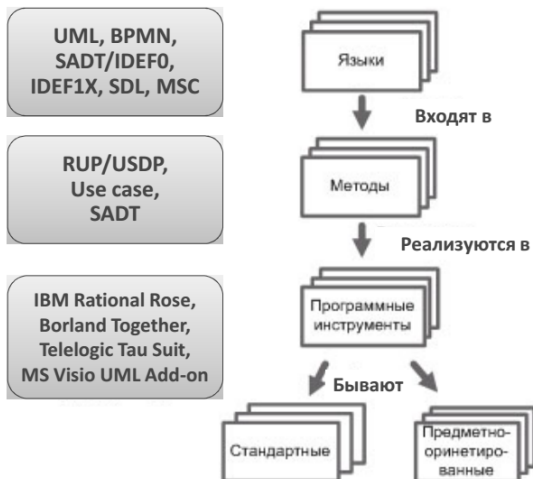
- диаграммы предметной области;
- диаграммы предполагаемого дизайна;
- диаграммы завершенной реализации;

Три уровня:

- концептуальный уровень;
- уровень спецификации;
- уровень реализации.

# Визуальное моделирование и его средства

Визуальное моделирование применяется на практике с помощью методов, языков и соответствующих программных инструментов.

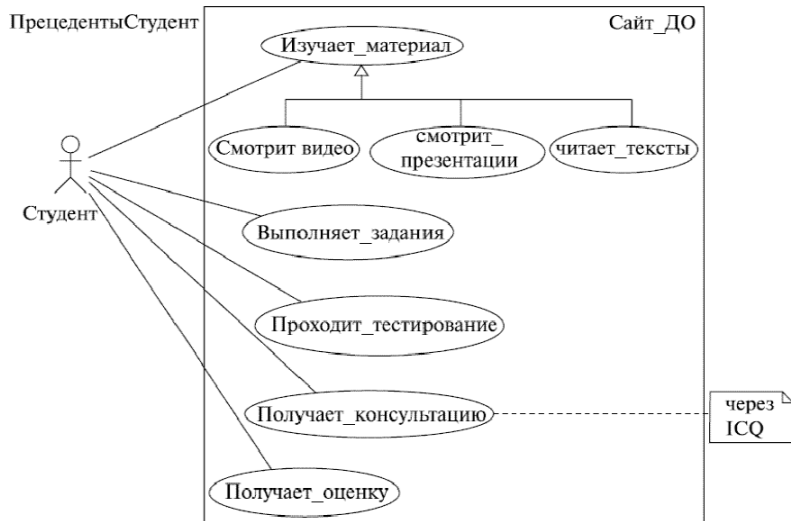


# Язык UML

## Три вида диаграмм UML:

- статические - описывают неизменную логическую структуру программы, а именно элементы - классы, объекты, структуры данных - и отношения между ними;
- динамические - показано, как программные сущности изменяются во время выполнения: поток выполнения или изменение состояния сущностей;
- физические - изображается неизменная физическая структура системы: исходные файлы, библиотеки, двоичные файлы, файлы данных и прочее, а также связи между ними.

# Понятие и применение UML



# Понятие и применение UML

**UML (Unified Modeling Language)** — унифицированный язык моделирования для описания, визуализации и документирования объектно-ориентированных систем в процессе их анализа и проектирования





# Назначение UML

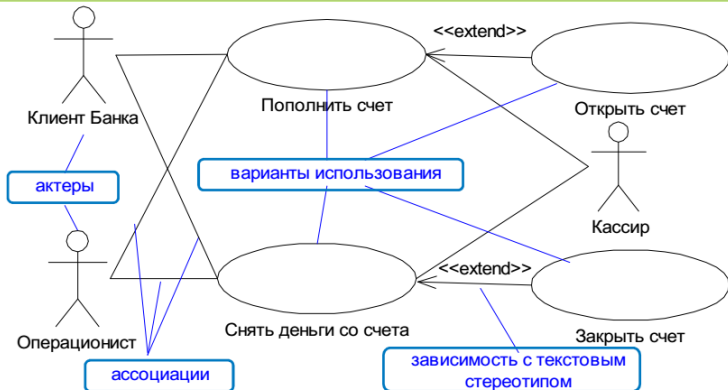
- Предоставить разработчикам **легко воспринимаемый и выразительный язык визуального моделирования**, специально **предназначенный для разработки и документирования моделей сложных систем** различного целевого назначения.
- Снабдить исходные понятия языка UML **возможностью расширения и специализации для более точного представления моделей систем в конкретной предметной области**.
- Графическое представление моделей в нотации UML **не должно зависеть от конкретных языков программирования и инструментальных средств проектирования**.
- **Облегчение контроля хода разработки** благодаря повышению уровня абстракции.
- **Облегчение взаимодействия** между разработчиками ПО и представителями других специальностей.

# Виды диаграмм UML



# Диаграмма вариантов использования

**Диаграмма вариантов использования** - диаграмма, на которой изображаются варианты использования проектируемой системы, заключенные в границу системы, и внешние актеры, а также определенные отношения между актерами и вариантами использования



# Диаграмма вариантов использования

## Назначение диаграммы вариантов использования

Определить общие **границы функциональности** проектируемой системы в контексте моделируемой **предметной области**

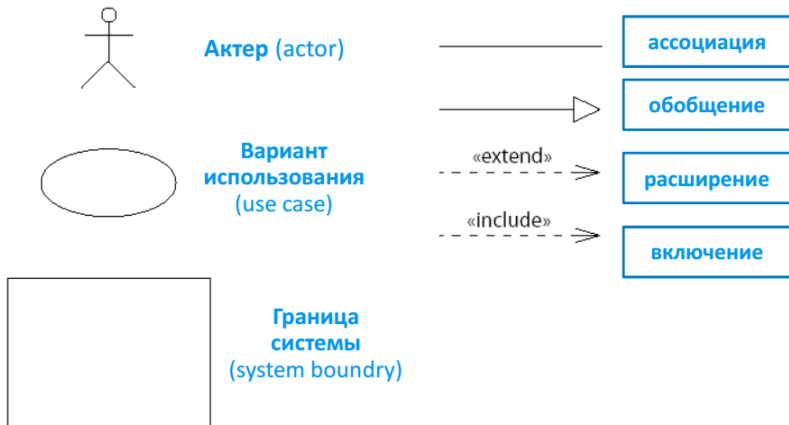
Специфицировать **требования к функциональному поведению** проектируемой системы в **форме вариантов использования**

Разработать исходную **концептуальную модель системы** для ее последующей детализации в форме **логических и физических моделей**.

Подготовить исходную **документацию** для взаимодействия **разработчиков системы с ее заказчиками и пользователями**

# Диаграмма вариантов использования

## Основные обозначения на диаграмме вариантов использования



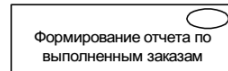
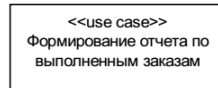
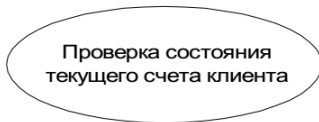
# Диаграмма вариантов использования

## Вариант использования (Use Case)

– представляет собой общую спецификацию совокупности выполняемых системой действий с целью предоставления некоторого наблюдаемого результата, который имеет значение для одного или нескольких актеров

Отвечает на вопрос «Что должна выполнять система?», не отвечая на вопрос «Как она должна выполнять это?»

Имена – отглагольное существительное или глагол в неопределенной форме

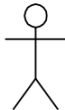


# Диаграмма вариантов использования

## Актер (actor)

- любая внешняя по отношению к проектируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач

*Примеры актеров:* кассир, клиент банка, банковский служащий, президент, продавец магазина, менеджер отдела продаж, пассажир авиарейса, водитель автомобиля, администратор гостиницы, сотовый телефон



Клиент банка

`<<actor>>`  
**Посетитель  
Интернет-магазина**



Удаленный  
пользователь

# Диаграмма вариантов использования

## Отношения на диаграмме вариантов использования

<i>Relationship</i>	<i>Function</i>	<i>Notation</i>
ассоциация	Служит только для обозначения взаимодействия актера с вариантом использования.	_____
включение	Специфицирует тот факт, что некоторый вариант использования содержит поведение, определенное в другом варианте использования	«include» - - - - ➔
расширение	Определяет взаимосвязь одного варианта использования с другим, функциональность или поведение которого задействуется первым не всегда, а только при выполнении некоторых дополнительных условий	«extend» - - - - ➔
обобщение	Предназначено для спецификаций того факта, что один элемент модели является специальным или частным случаем другого элемента модели	—————>



# Диаграмма вариантов использования

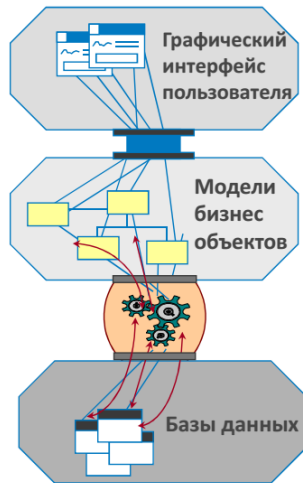
## Достоинства модели вариантов использования :

- Определяет пользователей и границы системы;
- Определяет системный интерфейс;
- Удобна для общения пользователей с разработчиками;
- Используется для написания тестов;
- Является основой для написания пользовательской документации;
- Хорошо вписывается в любые методы проектирования (как объектно-ориентированные, так и структурные).

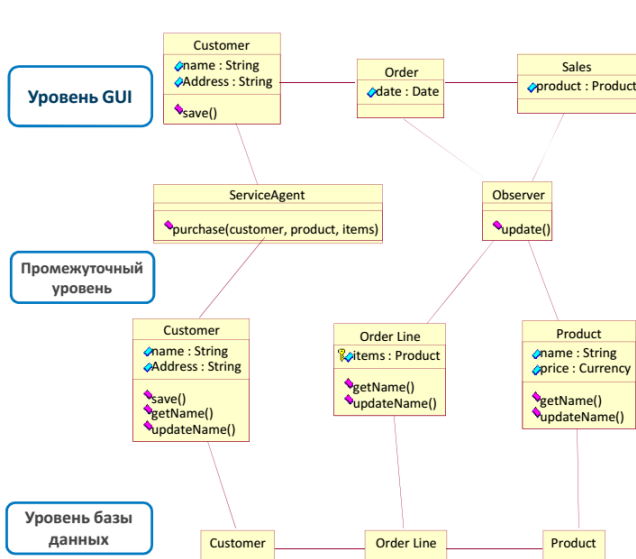
# Диаграмма классов

**Проблема:**

Как  
представить  
архитектуру  
программной  
системы?



# Диаграмма классов



Архитектура  
программной  
системы в  
нотации UML

# Диаграмма классов

## Диаграмма классов — основная логическая модель проектируемой системы

- **Диаграмма классов (*class diagram*)** — диаграмма, предназначенная для представления модели статической структуры программной системы в терминологии классов объектно-ориентированного программирования.
- Диаграмма классов представляет собой **граф, вершинами или узлами которого являются элементы типа “классификатор”**, которые **связаны различными типами структурных отношений**.
- **Классификатор (*classifier*)** — специальное понятие, предназначенное для классификации экземпляров, которые имеют общие характеристики.

# Диаграмма классов

## Характеристика (feature)

Понятие, предназначенное для спецификации особенностей структуры и поведения экземпляров классификаторов.

## Структурная характеристика (structural feature)

Типизированная характеристика классификатора, которая специфицирует структуру его экземпляров.

## Характеристика поведения (behavioral feature)

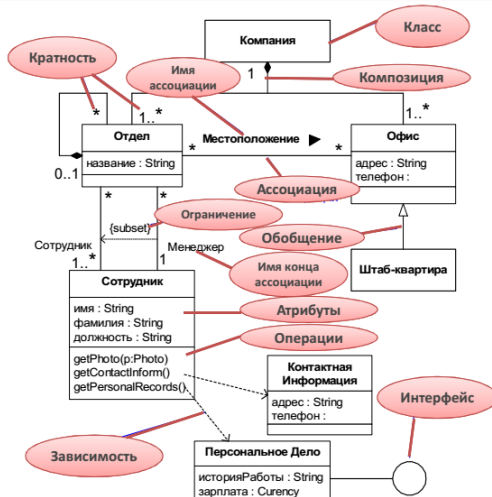
Характеристика классификатора, которая специфицирует некоторый аспект поведения его экземпляров.

## Класс (class)

Элемент модели, который описывает множество объектов, имеющих одинаковые спецификации характеристик, ограничений и семантики.

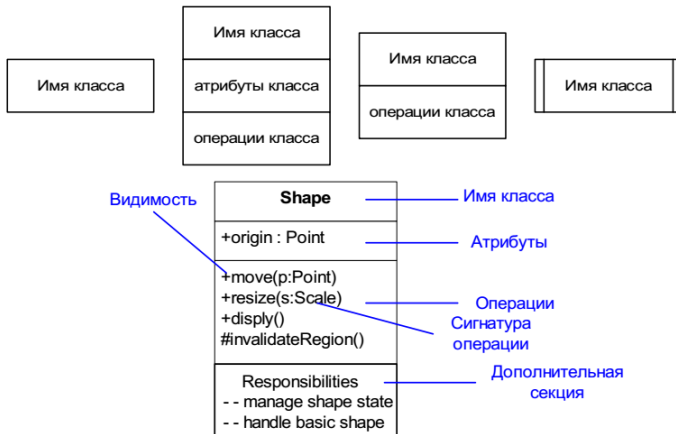
# Диаграмма классов

## Основные обозначения на диаграмме



# Диаграмма классов

## Варианты графического изображения класса на диаграмме классов



# Диаграмма классов

## Разновидности классов

### Абстрактный (abstract)

класс **не имеет экземпляров или объектов**, для обозначения его имени используется наклонный шрифт

### Активный класс (active class)

класс, каждый экземпляр которого **имеет свою собственную нить управления**

### Пассивный класс (passive class)

класс, каждый экземпляр которого **выполняется в контексте некоторого другого объекта**

**Квалифицированное имя (qualified name)** используется для того, чтобы явно указать, к какому пакету относится тот или иной класс. Для этого применяется специальный символ в качестве разделителя имени – двойное двоеточие “::”.

Имя класса без символа разделителя называется **простым именем** класса.



# Диаграмма классов

## Атрибут (attribute) класса

- служит для представления отдельной структурной характеристики или свойства, которое является общим для всех объектов данного класса.
- $\langle \text{атрибут} \rangle ::= [\langle \text{видимость} \rangle] [\text{'/'}] \langle \text{имя} \rangle [\text{'.'}]$   
 $\langle \text{тип атрибута} \rangle [\text{'['} \langle \text{кратность} \rangle \text{'}] [\text{'='} \langle \text{значение по умолчанию} \rangle]$   
 $[\text{'{'} \langle \text{модификатор атрибута} \rangle [\text{','} \langle \text{модификатор атрибута} \rangle]^* \text{'}}]$   
где:

$\langle \text{видимость} \rangle ::= \text{'+'} \mid \text{'-'} \mid \text{'\#'} \mid \text{'\sim'}$ .

- видимость (visibility)** может принимать одно из 4-х возможных значений и отображаться либо посредством специального символа, либо соответствующего ключевого слова.

# Диаграмма классов

## Вид видимости

**+ public**  
(общедоступный)

Общедоступный элемент является видимым всеми элементами, который имеют доступ к содержимому пространства имен, который им владеет.

**- private**  
(закрытый)

Закрытый элемент является видимым только внутри пространства имен, который им владеет.

**# protected**  
(защищенный)







Защищенный элемент является видимым для элементов, которые имеют отношение обобщения с пространством имен, который им владеет.

**~ package**  
(пакет)

Элемент, помеченный как имеющий пакетную видимость, является видимым всеми элементами в ближайшем охватывающем пакете в предположении. За пределами ближайшего охватывающего пакета элемент, помеченный как имеющий пакетную видимость, не является видимым.

# Диаграмма классов

## Отношения на диаграмме классов

association	Представление произвольного отношения между экземплярами классов	
generalization	Отношение типа "Общее-Частное", обладающая свойством наследования свойств	
aggregation	Отношение типа "Часть-Целое"	
composition	Более сильная форма отношения типа "Часть-Целое"	
realization	Отношение между спецификацией и ее выполнением	
dependency	Направленное отношение между двумя элементами модели с открытой семантикой	

# Диаграмма классов

**Ассоциация (*association*)**  
произвольное отношение или  
взаимосвязь между классами

**Имя конца ассоциации**  
специфицирует *роль* (role), которую играет класс, расположенный на соответствующем конце рассматриваемой ассоциации.

**Символ отсутствия навигации**  
**(non navigable)**  
изображается с помощью буквы «X» на линии у конца ассоциации

**Кратность конца ассоциации**  
специфицирует возможное количество экземпляров соответствующего класса, которое может соотноситься с одним экземпляром класса на другом конце этой ассоциации

**Видимость конца ассоциации**  
специфицирует возможность доступа к соответствующему концу ассоциации с других ее концов

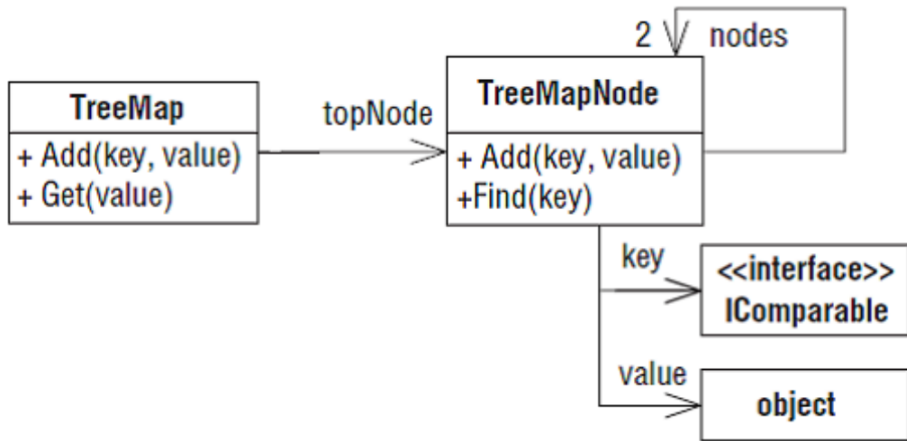
**Символ наличия навигации**  
**(navigable)**  
изображается с помощью простой стрелки в форме буквы «V» на конце ассоциации

# Диаграмма классов

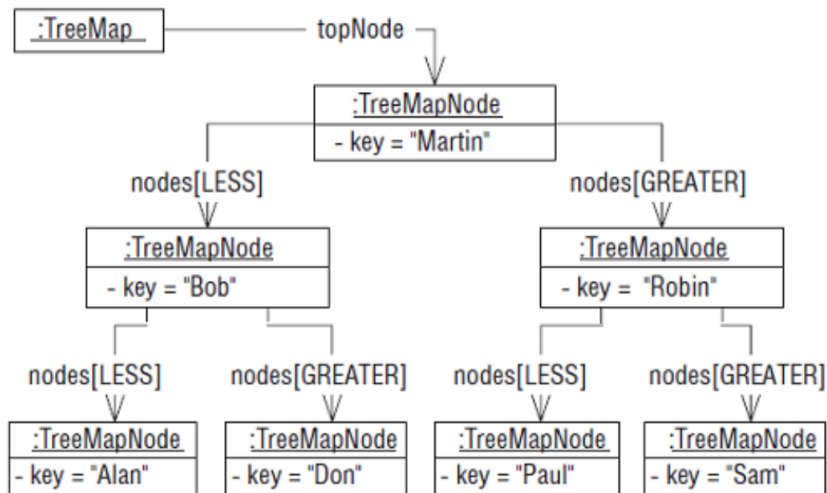
## Общие рекомендации по изображению диаграмм в нотации языка UML

- Каждая диаграмма должна служить **законченным представлением** соответствующего **фрагмента моделируемой предметной области**.
- Все **сущности** на диаграмме модели должны быть **одного концептуального уровня**.
- **Вся информация** о сущностях должна быть **явно представлена** на диаграммах.
- Диаграммы **не должны содержать противоречивой информации**.
- Диаграммы **не следует перегружать текстовой информацией**.
- Каждая диаграмма должна быть **самодостаточной для правильной интерпретации** всех ее элементов и **понимания семантики** всех используемых графических символов.

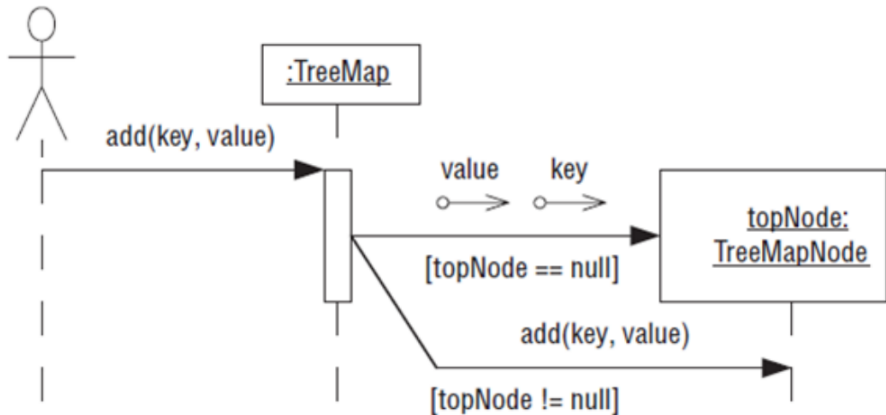
# Диаграмма классов



# Диаграмма объектов

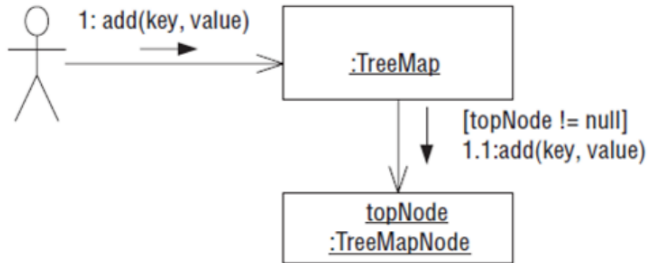


# Диаграмма последовательности

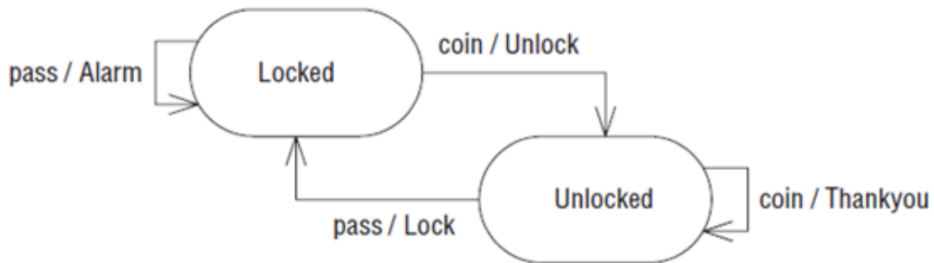




# Диаграмма взаимодействия



# Диаграмма состояний



# Эффективное использование UML

## Эффективное использование UML

- общение с другими людьми;
- создание карт крупных программных структур;
- создание финальной документации;
- возможность итеративного уточнения;
- схематическое представление программного кода.

Зачем строить модели программ? Нужно ли проектировать систему до конца, прежде чем приступить к кодированию?

# Эффективное использование UML

## Когда рисовать диаграммы

- несколько человек должны понимать структуру конкретной части дизайна, поскольку будут работать над ней одновременно;
- необходимо добиться консенсуса, но хотя бы два человека не согласны с конкретным элементом дизайна;
- возникает потребность мысленно визуализировать идею дизайна, и диаграммы могут помочь в ее обдумывании;
- требуется объяснить структуру части кода самому себе или кому-то еще;
- близится конец проекта, и заказчик попросил представить диаграммы как часть документации для третьих лиц.

# Эффективное использование UML

## Использование CASE-средств

- Разве CASE-средства не помогают рисовать UML-диаграммы?
- Разве CASE-средства не упрощают совместную работу над диаграммами в больших командах?
- Разве CASE-средства не упрощают генерацию кода?
- Как насчет CASE-средств, которые одновременно являются IDE и показывают код и диаграммы вместе?