# Анализ и моделирование на UML

Направление подготовки
“Информационные системы и технологии”

Максим Валерьевич Хлопотов,
старший преподаватель кафедры ИС

# Темы лекционных занятий

1. Введение в UML
2. Моделирование использования
3. **Моделирование структуры**
4. Моделирование поведения
5. Дисциплина моделирования

# Моделирование структуры

- Моделируя структуру, мы описываем составные части системы и отношения между ними.
- UML является объектно-ориентированным языком моделирования, поэтому не удивительно, что основным видом составных частей, из которых состоит система, являются объекты.

# Диаграммы классов

- Диаграмма классов является основным средством моделирования структуры UML.
- Диаграммы классов наиболее информационно насыщены по сравнению с другими типами канонических диаграмм UML.
- На диаграммах классов в качестве сущностей применяются прежде всего классы, как в своей наиболее общей форме, так и в форме многочисленных стереотипов и частных случаев: интерфейсы, типы данных, процессы и др.
- Кроме того, в диаграмме классов могут использоваться (как и везде) пакеты и примечания.

# Интерфейс

- В UML имеется несколько частных случаев классификаторов, которые, подобно классам, предназначены для моделирования структуры, но обладают рядом специфических особенностей. Наиболее важным из них являются интерфейсы.
- Интерфейс — это именованный набор абстрактных операций.
- Другими словами, интерфейс — это абстрактный класс, в котором нет атрибутов и все операции абстрактны. Поскольку интерфейс — это абстрактный класс, он не может иметь непосредственных экземпляров.

# Интерфейс

- Между интерфейсами и другими классификаторами, в частности классами, на диаграмме классов применяются два отношения:
- классификатор (в частности, класс) использует интерфейс — это показывается с помощью зависимости со стереотипом «call»;
- классификатор (в частности, класс) реализует интерфейс — это показывается с помощью отношения реализации.

# Интерфейс

- Между интерфейсами и другими классификаторами, в частности классами, на диаграмме классов применяются два отношения:
- классификатор (в частности, класс) использует интерфейс — это показывается с помощью зависимости со стереотипом «call»;
- классификатор (в частности, класс) реализует интерфейс — это показывается с помощью отношения реализации.

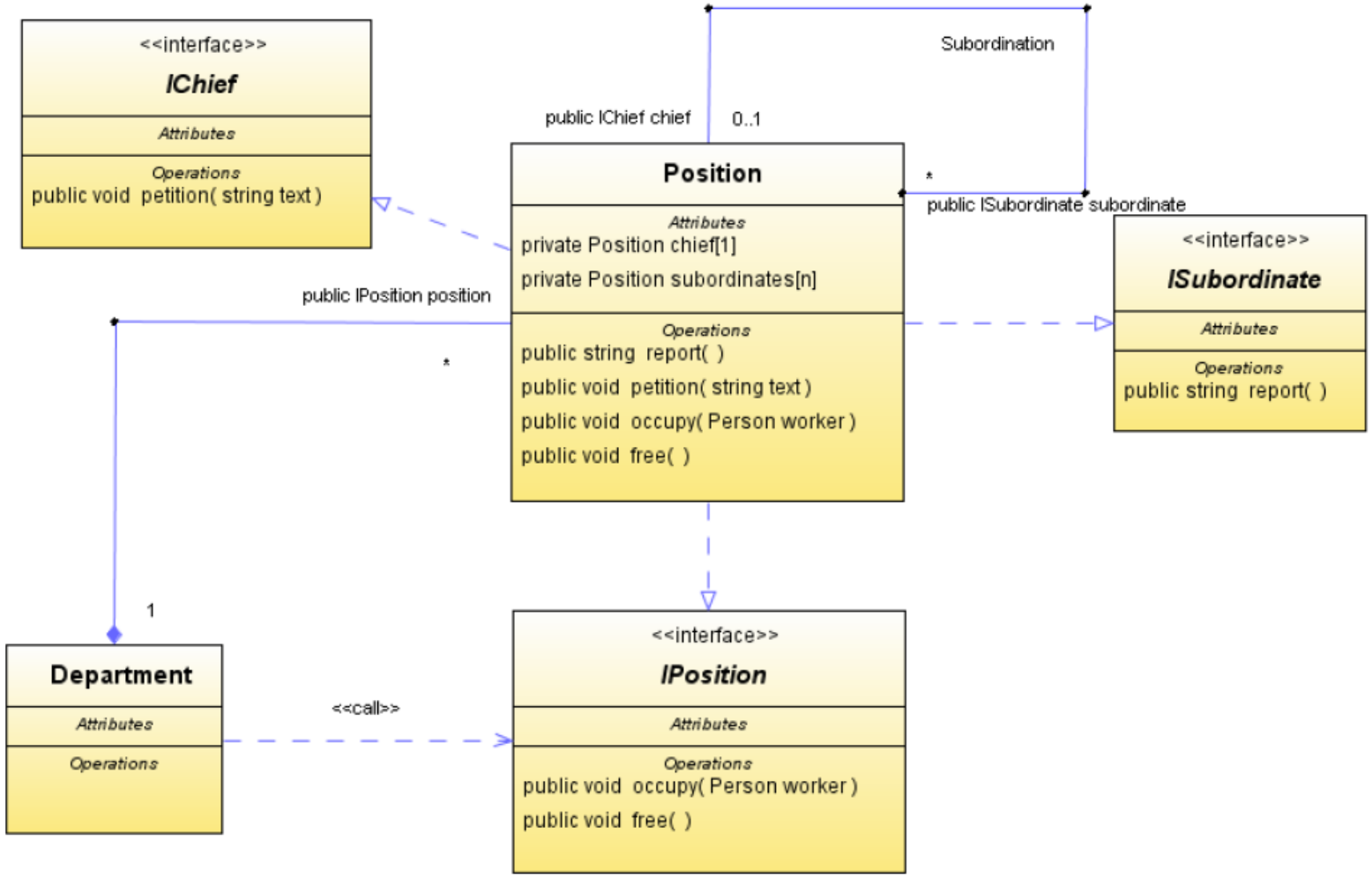
# Интерфейс

- Роль — это интерфейс, который предоставляет классификатор в данной ассоциации.

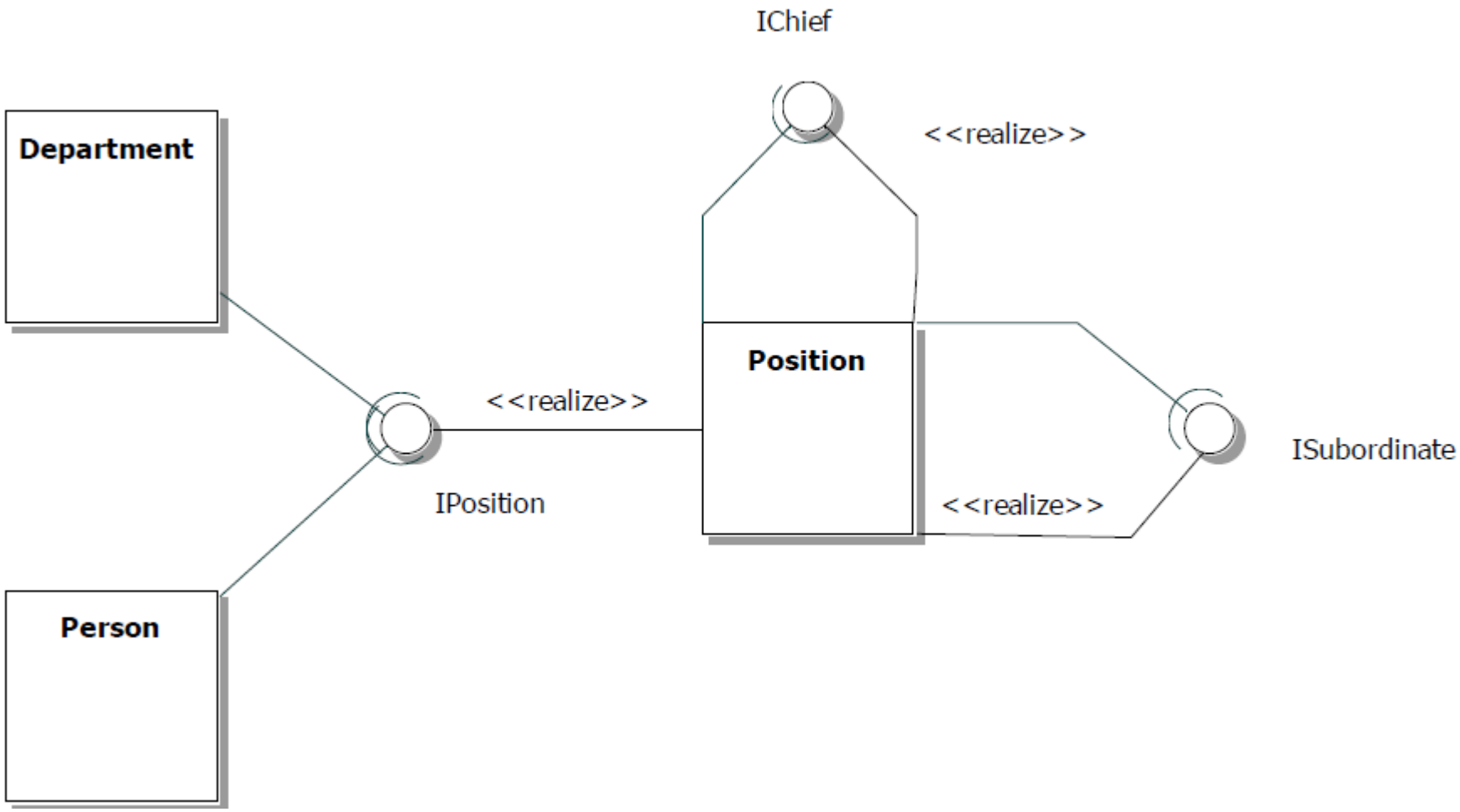
# Интерфейс

- Допустим, что класс Department для реализации операций связанных с движением кадров, использует операции класса Position, позволяющие занимать и освобождать должность — другие операции класса Position классу Department не нужны. Для этого можно определить соответствующий интерфейс IPosition и связать его отношениями с данными классами.

# Интерфейс


<<interface>>
IChief
Attributes
Operations
public void petition( string text )
Subordination
public IChief chief
0..1
Position
Attributes
private Position chief[1]
private Position subordinates[n]
Operations
public string report( )
public void petition( string text )
public void occupy( Person worker )
public void free( )
*
public ISubordinate subordinate
<<interface>>
ISubordinate
Attributes
Operations
public string report( )
public IPosition position
*
1
Department
Attributes
Operations
<<call>>
<<interface>>
IPosition
Attributes
Operations
public void occupy( Person worker )
public void free( )

# Интерфейс


IChief
<<realize>>
Department
Position
<<realize>>
ISubordinate
IPosition
<<realize>>
Person

# Типы данных

- *Тип данных* — это совокупность  множества значений (может быть очень большого или даже потенциально бесконечного) и конечного множества операций, применимых к данным значениям.
- UML не является сильно типизированным языком: например, в модели можно указывать типы атрибутов классов и параметров операций, но это не обязательно.

# Типы данных

- Для каких элементов модели можно указать тип?
- Что можно использовать в качестве указания типа?

# Типы данных

- Для каких элементов модели можно указать тип?
- В UML типизированы могут быть:
- атрибуты классов, в том числе классов ассоциаций;
- параметры операций, в том числе тип возвращаемого значения;
- роли полюсов ассоциаций;
- параметры шаблонов (см. ниже).

# Типы данных

- В модели UML можно использовать три вида типов данных:
- Примитивные типы, которые считаются предопределенными в UML
- Типы данных, которые определены в языке программирования, поддерживаемым инструментом
- Типы данных, которые определены в модели

# Типы данных

- Примитивные типы, которые считаются предопределенными в UML — таковых, как минимум, три: строка, целое число и значение даты/времени. Инструменты вправе расширять этот набор и использовать подходящие названия.
- Типы данных, которые определены в языке программирования, поддерживаемым инструментом. Это могут быть как названия встроенных типов, так и сколь угодно сложные выражения, доставляющие тип, если таковые допускаются языком.
- Типы данных, которые определены в модели. В стандарте UML предусмотрен только один конструктор типов данных: перечислимый тип, который определяется с помощью стереотипа «enumeration». Наряду со стандартным стереотипом «enumeration» многие инструменты допускают использование стереотипа «datatype», который означает построение типа данных с помощью не специфицированного конструктора типов.

# Шаблон

- Еще одной сущностью, специфической для диаграмм классов, являются шаблоны.
- *Шаблон — это класс с параметрами.* Параметром может быть любой элемент описания класса — тип составляющей, кратность атрибута и т. д. На диаграмме шаблон изображается с помощью прямоугольника класса, к которому в правом верхнем углу присоединен пунктирный прямоугольник с параметрами шаблона.
- Описания параметров перечисляются в этом прямоугольнике через запятую. Описание каждого параметра имеет вид:
- ИМЯ : тип

# Шаблон

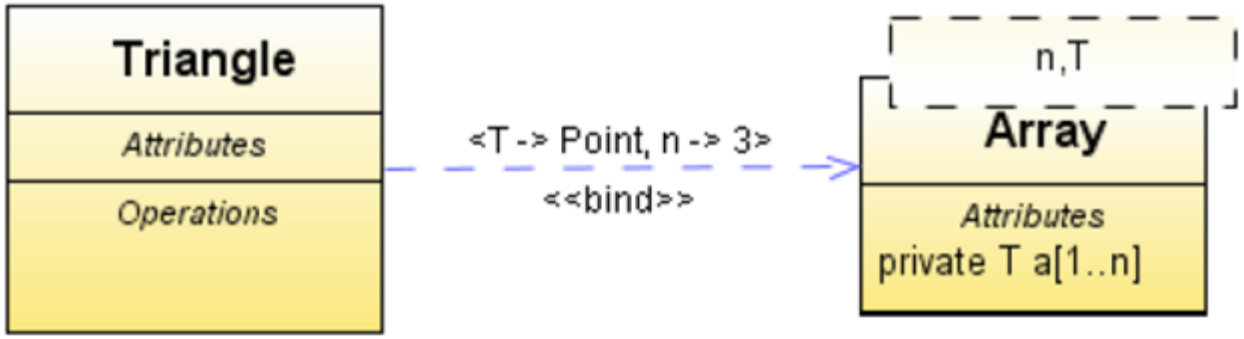
- Назначение и область применения шаблонов.
- Шаблоны нужны, чтобы определить некоторую общую параметрическую конструкцию класса один раз, и затем использовать ее многократно, подставляя конкретные значения аргументов.

# Шаблон

- Сам по себе шаблон не может непосредственно использоваться в модели. Для того, чтобы на основе шаблона получить конкретный класс, который может использоваться в модели, нужно указать явные значения аргументов. Такое указание называется *связыванием. В UML применяются два способа связывания:*
- *явное связывание* — зависимость со стереотипом «bind», в которой указаны значения аргументов;
- *неявное связывание* — определение класса, имя которого имеет формат имя_шаблона < аргументы >
- Явное связывание более наглядно, неявное связывание менее наглядно, зато записывается короче.

# Шаблон

- Здесь определен шаблон Array, имеющий два параметра: n и T. Этот шаблон применяется для создания массивов определенной длины, содержащих элементы определенного типа. В данном случае с помощью явного связывания определен класс Triangle как массив из трех элементов типа Point.
- *На следующем занятии будет тест, на котором я задам вопрос: «Показать то же самое с помощью неявного связывания».*



# Диаграммы классов

- Диаграммы классов содержат множество деталей. Для практически значимых систем диаграммы классов в конечном итоге получаются довольно сложными. Пытаться прорисовать сложную диаграмму классов сразу "на всю глубину" нерационально — слишком велик риск "утонуть" в деталях.
- Удачная модель структуры сложной системы создается за несколько (может быть даже за несколько десятков) итераций, в которых моделирование структуры перемежается моделированием поведения.

# Диаграммы классов

- Описывать структуру удобнее параллельно с описанием поведения. Каждая итерация должна быть небольшим уточнением как структуры, так и поведения.
- Не обязательно включать в модель все классы сразу. На первых итерациях достаточно идентифицировать очень небольшую (10%) долю всех классов системы.
- Не обязательно определять все свойства класса сразу. Начните с имени — операции и атрибуты постепенно выявятся в процессе моделирования поведения.
- Не обязательно показывать на диаграмме все свойства класса. В процессе работы диаграмма должна легко охватываться одним взглядом.
- Не обязательно определять все отношения между классами сразу. Пусть класс на диаграмме "висит в воздухе" — ничего с ним не случится.

# Выводы

- Диаграммы классов моделируют структуру объектов и связей между ними.
- Классы выбираются на основе анализа предметной области, взаимного согласования элементов модели и общих теоретических соображений.
- Сущности на диаграммах классов связываются главным образом отношениями ***ассоциации*** (в том числе агрегирования и композиции) и ***обобщения***.
- Базовая нотация ассоциации позволяет указать, что объекты ассоциированных классов могут взаимодействовать во время выполнения. Для ассоциации в UML предусмотрено наибольшее количество различных дополнений.

# Диаграммы реализации

- Диаграммы реализации — это обобщающее название для диаграмм компонентов и диаграмм размещения. Название объясняется тем, что данные диаграммы приобретают особую важность на позднейших фазах разработки — на фазах реализации и перехода. На ранних фазах разработки — анализа и проектирования — эти диаграммы либо вообще не используются, либо имеют самый общий, не детализированный вид.

# Диаграммы реализации

- Диаграммы компонентов и размещения имеют больше общего, чем различий, поскольку предназначены для моделирования двух теснейшим образом связанных структур:
- • структуры компонентов в приложении;
- • структуры используемых вычислительных ресурсов.

# Диаграммы компонентов

- Диаграмма компонентов предназначена для перечисления и указания взаимосвязей артефактов моделируемой системы.
- На диаграмме компонентов применяются следующие основные типы сущностей:
  - компоненты;
  - интерфейсы;
  - классы;
  - объекты.
- На диаграмме компонентов обычно используются отношения следующих типов:
  - зависимость;
  - ассоциация (главным образом в форме композиции);
  - реализация.

# Компонент

- Абсолютно формальное определение компонента в UML дать невозможно.
- Компонент — это физически существующий и заменяемый артефакт системы.
- Прежде всего, это компонент в смысле сборочного программирования: особым образом оформленная часть программного кода, которая может работать в составе более сложной программы.
- Компоненты понимаются в UML в наиболее общем смысле: это не только исполнимые файлы с кодами программы, но и исходные тексты программ, веб-страницы, справочные файлы, сопроводительные документы, файлы с данными и вообще любые артефакты, которые тем или иным способом используются при работе приложения и входят в его состав.

# Компонент

- При выделении компонентов применяются следующие неформальные критерии.
  - Компонент нетривиален. Это нечто более сложное и объемное, чем фрагмент кода или одиночный класс.
  - Компонент независим, но не самодостаточен. Он содержит все, что нужно для функционирования, но предназначен для работы во взаимодействии с другими компонентами.
  - Компонент однороден. Он выполняет несколько взаимосвязанных функций, которые могут быть естественным образом охарактеризованы как единое целое в контексте более сложной системы.
  - Компонент заменяем. Он поддерживает строго определенный набор интерфейсов и может быть без ущерба для функционирования системы заменен другим компонентом, поддерживающим те же интерфейсы.

# Диаграмма компонентов

- В случае разработки "монолитного" настольного приложения диаграмма компонентов не нужна — она оказывается тривиальной и никакой полезной информации не содержит.
- Таким образом, диаграммы компонентов применяются только при моделировании многокомпонентных приложений.

# Диаграмма компонентов

- Если приложение поставляется в виде "конструктора" (набора "кубиков" — компонентов) из которого при установке собирается конкретный уникальный экземпляр приложения, то диаграммы компонентов оказываются просто незаменимым средством.
- Многие современные приложения поставляются в виде большого (десятки и сотни) набора компонентов, из которых "на месте" собирается нужная пользователю, часто уникальная конфигурация.
- Рекомендуют использовать диаграммы компонентов для управления конфигурацией не только на фазе поставки и установки программного обеспечения, но и в процессе разработки: для отслеживания версий компонентов, вариантов сборки и т. п.

# Диаграмма компонентов



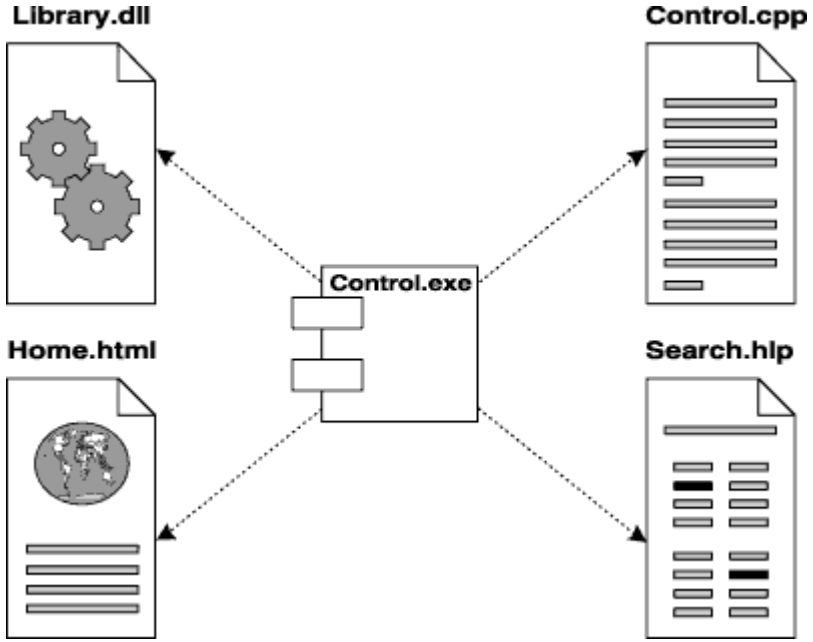
Диаграмма компонентов разрабатывается для следующих целей:

Визуализации общей структуры исходного кода программной системы.
Спецификации исполнимого варианта программной системы.
Обеспечения многократного использования отдельных фрагментов программного кода.
Представления концептуальной и физической схем баз данных.

# Стереотипы компонентов



исполнимый *компонент* Control .exe использует или импортирует некоторую функциональность *компонента* Library .dll, вызывает страницу гипертекста Home .html и файл помощи Search .hlp, а исходный текст этого исполнимого *компонента* хранится в файле Control .cpp.

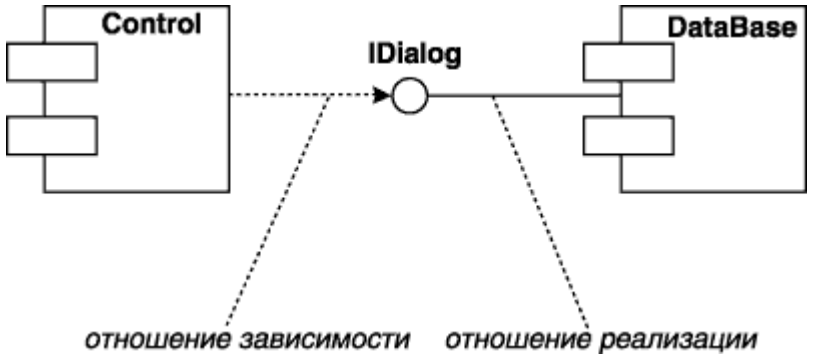


# Зависимость компонентов



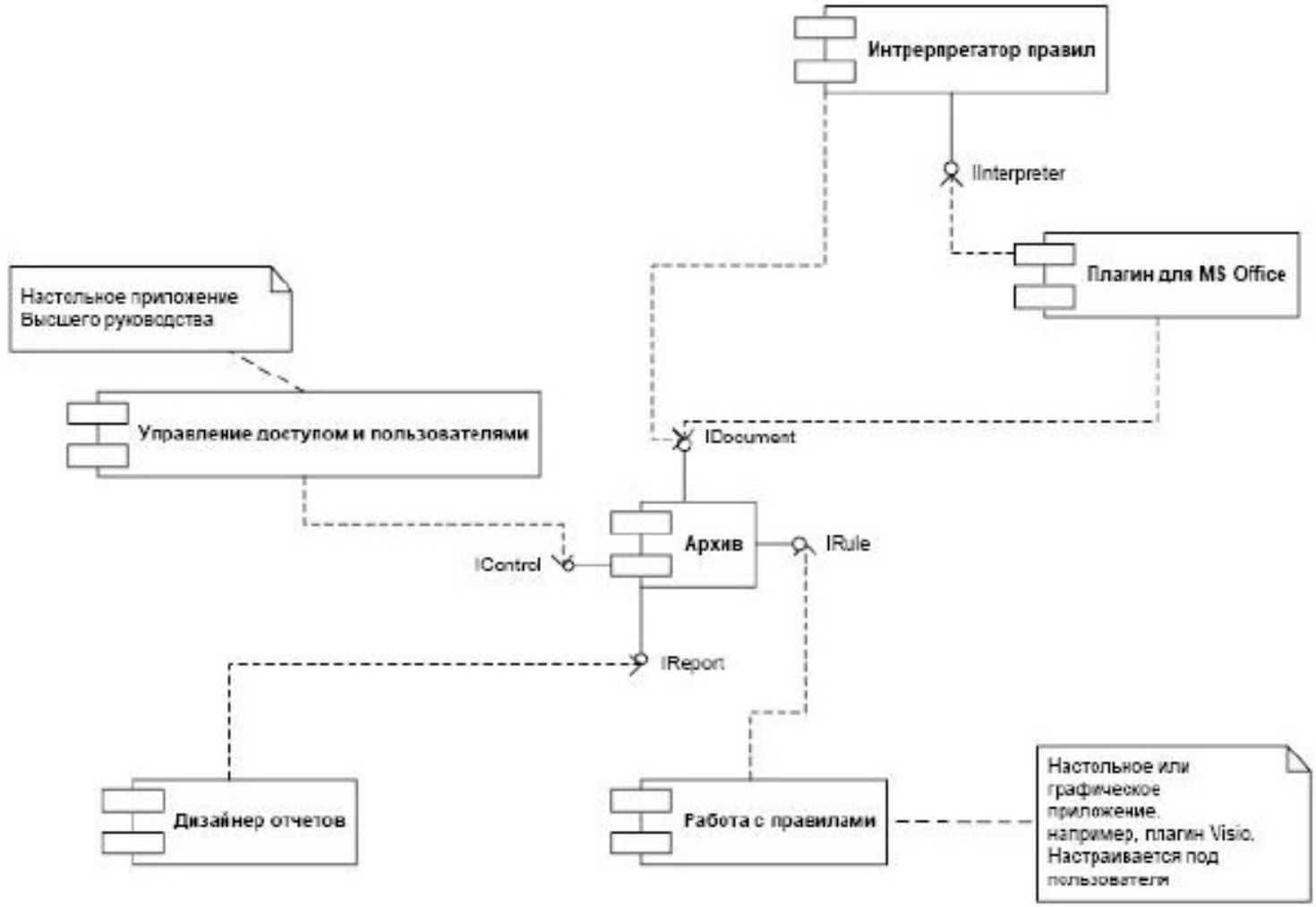
*компонент* с именем Control зависит от *импортируемого интерфейса* IDialog, который, в свою очередь, реализуется *компонентом* с именем DataBase.

При этом для второго *компонента* этот *интерфейс* является *экспортируемым*.



# Диаграмма компонентов





# Диаграмма размещения

- Последним структурным аспектом, который необходимо обсудить, является описание размещения компонентов относительно участвующих в работе вычислительных ресурсов.
- В UML для этой цели предназначены диаграммы размещения. В UML 2.0 эти диаграммы переименованы в диаграммы развертывания.
- Если речь идет о настольном приложении, которое целиком хранится и выполняется на одном компьютере, то отдельная диаграмма размещения не нужна.
- При моделировании  распределенных приложений значение диаграмм размещения резко возрастает.

# Диаграмма размещения

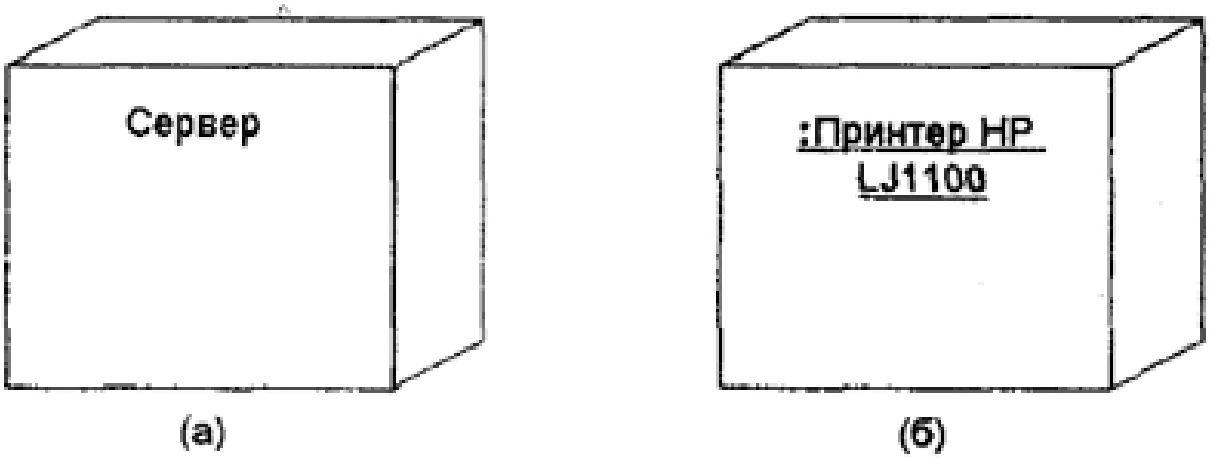
- На диаграмме размещения, по сравнению с диаграммами компонентов, применяются только один дополнительный тип сущности — узел и два дополнительных отношения: ассоциация между узлами и размещение компонента на узле.
- В остальном диаграммы размещения наследуют возможности диаграмм компонентов.

# Диаграмма размещения

- Узел — это физический вычислительный ресурс, участвующий в работе системы.
- Компоненты системы во время ее работы размещаются на узлах. В UML узел является классификатором, т. е. мы можем (и должны!) различать описание типа вычислительного ресурса (например, рабочая станция, последовательный порт) и описание экземпляра вычислительного устройства (например, устройство COM1 типа последовательный порт).
- Это различие моделируется согласно общему механизму UML: имя экземпляра узла подчеркивается, а имя типа узла — нет.

# Диаграмма размещения

- На диаграмме узел представляется фигурой, изображающей прямоугольный параллелепипед.
- На примере (а) - имя типа узла, (б) – имя экземпляра узла .

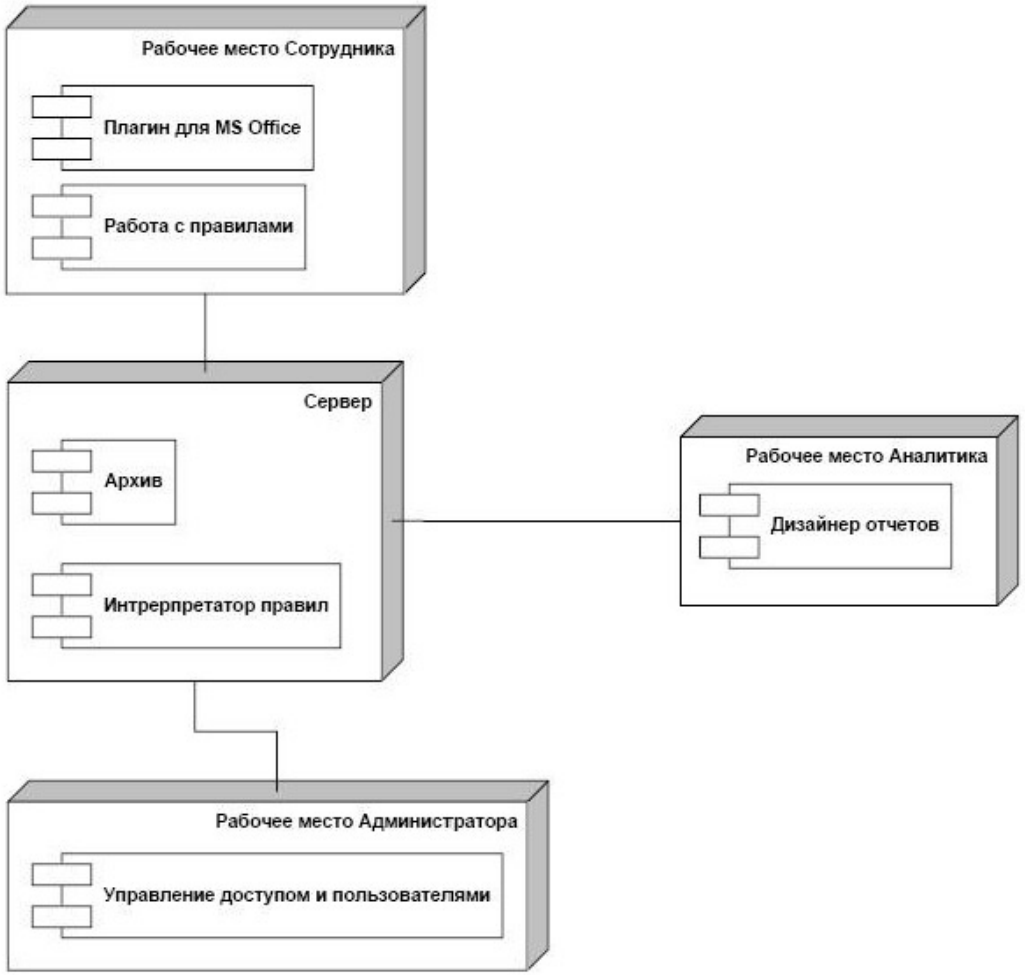


# Диаграмма размещения

- Ассоциация между узлами означает то же, что и в других контекстах: возможность обмена сообщениями.
- Применительно к вычислительным сетям ассоциация означает наличие канала связи. Если нужно указать дополнительную информацию о свойствах канала, то это можно сделать используя общие механизмы: стереотипы, ограничения и именованные значения, приписанные ассоциации.

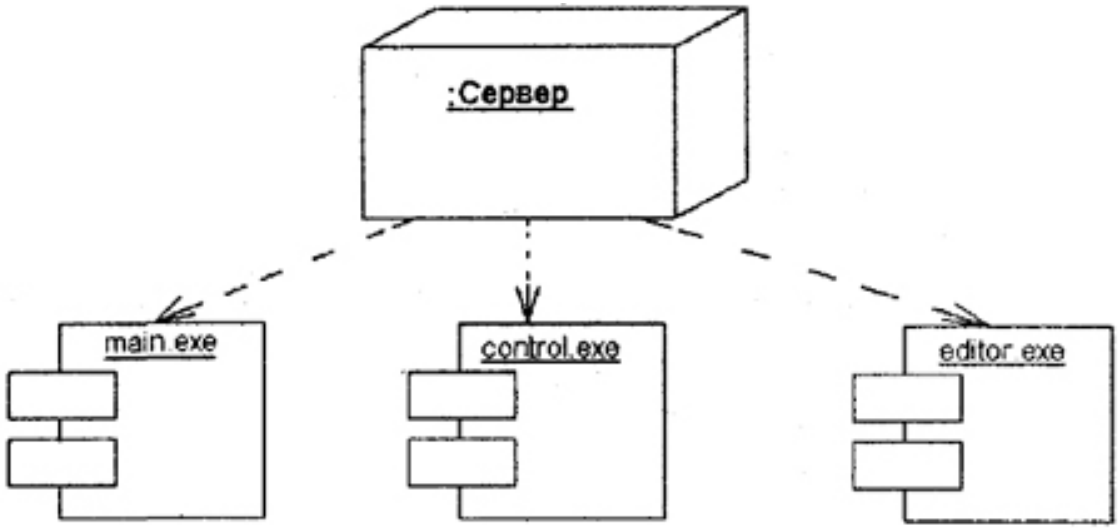
# Диаграмма размещения

- Размещение компонента на узле, как правило, изображают, помещая фигуру компонента внутрь фигуры узла.



# Диаграмма размещения

- Если это по каким-либо причинам неудобно, то отношение размещения можно передать отношением зависимости от узла к компоненту.



# Порядок выполнения л.р.2

- 1. Составить словарь предметной области.
- 2*. Составить перечень используемых образцов проектирования
- 3. Используя методы идентификации классов, выделить классы.
- 4. Указать связи между классами.
- 5. Указать атрибуты классов.
- 6*. Указать операции классов.
- 7. Создать диаграммы реализации.

# Отчёт по л.р. 2

- Отчёт должен включать в себя:
- 1. Словарь предметной области
- 2. Перечень классов (с указанием атрибутов и операций)
- 3. Диаграммы классов (3 штуки)*
- 4. Диаграмма компонентов
- 5. Диаграмма развертывания

# Выводы

- Диаграммы компонентов моделируют структуру компонентов (артефактов) и взаимосвязей между ними.
- Диаграммы размещения моделируют структуру вычислительных ресурсов, а также размещение компонентов.