Диаграмма классов (class diagram)

Объект

- **Объект (object)** – это некоторая сущность реального мира.
- Объект обладает тремя характеристиками: **состоянием, поведением, индивидуальностью.**
- **Состояние (state)** определяется набором свойств, называемых атрибутами (attribute), и их значениями. *Студент имеет атрибуты номер зачетной книжки, фамилию, адрес.*
- **Поведение (behavior)** определяет что может делать сам объект и что могут делать другие с данным объектом. Поведение реализуется с помощью набора операций.

Класс

- Класс (class) – это описание группы объектов, обладающих общими свойствами, поведением, отношениями с другими объектами и семантикой.

  Курс лекций имеет название, номер. Его можно добавить, удалить. Его читает преподаватель и посещают студенты. Его должны посещать не более 10 и не менее 3 человек.

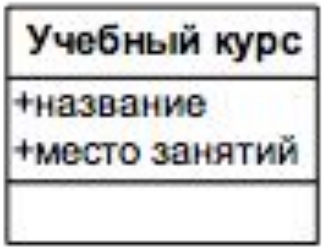
- Объект является экземпляром некоторого класса.
- Классы используются для составления словаря разрабатываемой системы. Это могут быть абстракции, являющиеся частью предметной области, либо классы, на которые опирается реализация. С их помощью описывают программные и аппаратные сущности.

Описание класса

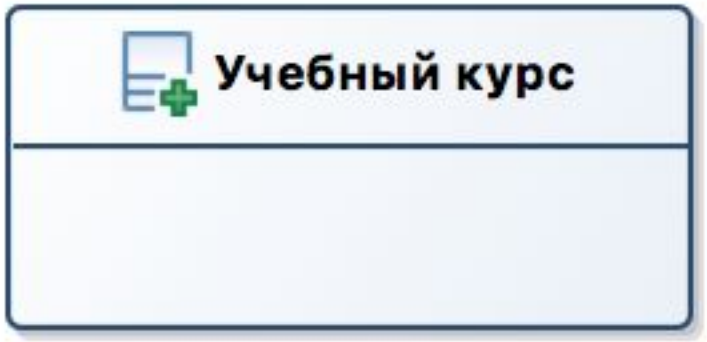
- ***Имя класса***. Оно должно быть уникальным внутри пакета, которому принадлежит класс.
- ***Атрибут*** – это свойство класса. Атрибут имеет имя и допустимое множества значений, т.е. значений, которые могут принимать экземпляры этого свойства.
  Класс может иметь любое число атрибутов или не иметь их вовсе.
- ***Операция*** – это то, что можно делать с объектом класса. У всех объектов класса имеется общий набор операций.

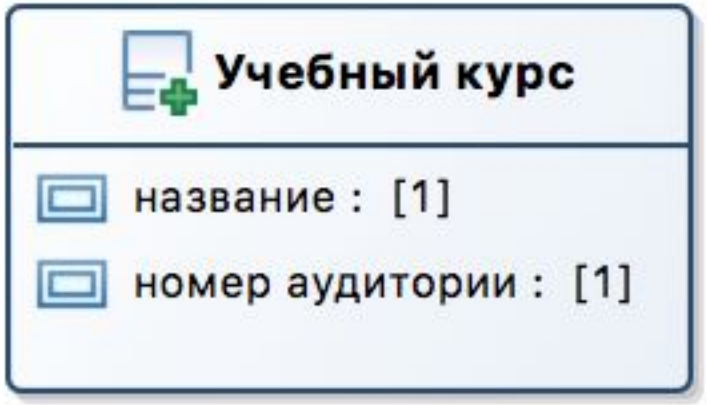
Графическое изображение класса

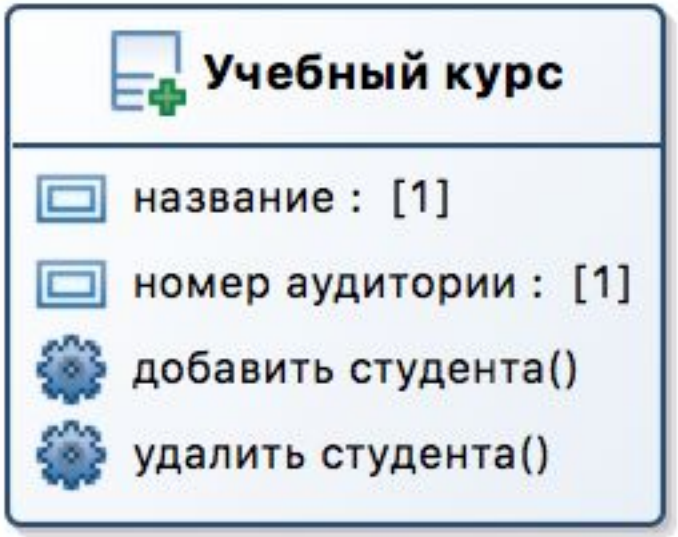
- Класс изображается в виде прямоугольника, разделенного на 3 части: имя, атрибуты, операции.
- На изображении класса всегда указывается имя. Остальные части могут отсутствовать.
- Не обязательно сразу показывать все его атрибуты и операции. Их может быть слишком много для одного рисунка.
- Класс обычно сворачивают, то есть изображают лишь некоторые из имеющихся атрибутов и операций, или не изображают их совсем.
- Пустой раздел в соответствующем месте прямоугольника может означать не отсутствие атрибутов или операций, а только то, что их не сочли нужным изобразить.



Изображение класса в UML Designer







Описание атрибутов (attributes)

- **Имя атрибута** – текстовая строка, которая используется для обозначения атрибута. Оно должно быть уникальным в пределах класса. Имя – единственный обязательный элемент при описании атрибута.

  фамилия_студента, название_курса
- **Тип** (type) определяет множество допустимых значений атрибута. Допустимые типы определяются языком программирования, который будет использоваться для реализации системы.

  string, float, int
- **Множественность** (multiplicity) – количество атрибутов данного типа.

  [0..1], [0..], [1..*], [1..5]*

  фамилия_имя_отчество [1..3]: string
- **Начальное значение** (initial value) – значение

Видимость атрибутов

- **Видимость** (visibility) определяет доступность атрибута из других классов. Возможные значения:

“+” – **общедоступный** (public) атрибут. Доступен из любого класса.

“-” – **закрытый** (private) атрибут. Недоступен для всех классов. Может использоваться только операциями данного класса.

“#” – **защищенный** (protected) атрибут. Недоступен для других классов, за исключением подклассов данного класса.ы

“~” – **пакетный** (package) атрибут. Доступен из любого класса, находящегося в одном пакете с данным классом или вложенном подпакете.

Описание операций (operations)

- **Имя операции** – текстовая строка, которая используется для обозначения операции. Оно должно быть уникальным в пределах класса. Имя – единственный обязательный элемент при описании операции.
- **Список параметров** (parameters) – набор параметров, которые использует операция. Параметр описывается именем, типом, видом. Вид параметра (kind) принимает одно из значений:
  - in – входной параметр
  - out – выходной параметр
  - inout – входной параметр, значение которого может быть изменено
- **Тип возвращаемого значения** (return type) – тип возвращаемого значения после выполнения операции.

Дополнительные свойства операций

- **Свойство параллельности** (call concurrency). Возможные значения свойства:
  - **последовательная** (sequential) – к операции должны обращаться последовательно. Если к операции обратятся несколько параллельных процессов, то система не гарантирует правильность выполнения операции.
  - **параллельная** (concurrent) – к операции могут одновременно обращаться несколько параллельных процессов
  - **охраняемая** (guarded) – если к операции будут обращаться несколько параллельных процессов, то операция будет выполнена только для одного из них, остальные будут блокированы.
- **Свойство полиморфизма** (Polymorphic) означает, что данная операция может быть переопределена в подклассе.

Видимость операций

- Видимость (visibility) определяет доступность операций из других классов. Возможные значения:

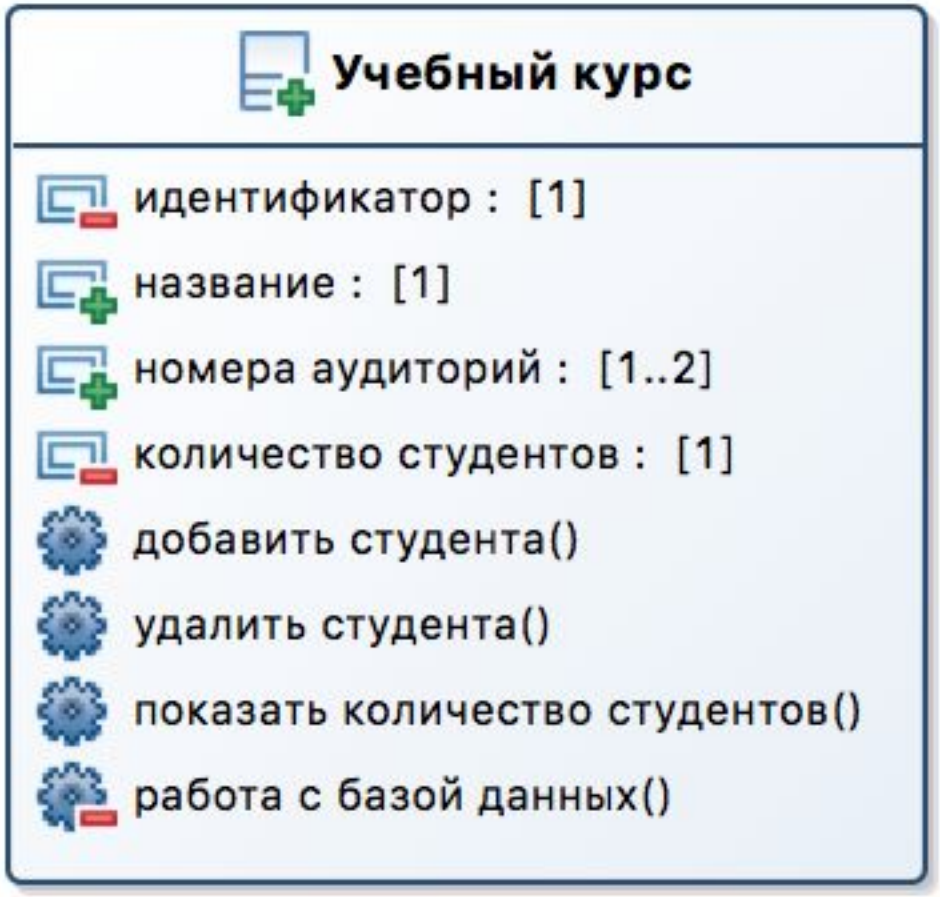
“+” – **общедоступная** (public) операция. Доступна из любого класса.

“-” – **закрытая** (private) операция. Недоступна для всех классов. Может использоваться только операциями данного класса.

“#” – **защищенная** (protected) операция. Недоступна для других классов, за исключением подклассов данного класса.

“~” – **пакетная** (package) операция. Доступна из любого класса, находящегося в одном пакете с данным классом или вложенном подпакете.

Пример класса



Отношения между классами (relationships)

- Отношение **зависимости** (dependency)
- Отношение **ассоциации** (association)
- Отношение **агрегации** (aggregation)
- Отношение **композиции** (composition)
- Отношение **обобщения** (generalization)

Отношение зависимости (dependency)

- Отношение зависимости используется, когда некоторое изменение одного элемента может потребовать изменения другого зависимого от него элемента модели. На диаграмме классов таким элементом является класс.
- Class_B является источником некоторой зависимости, а Class_A - клиентом этой зависимости.
- Один класс может зависеть сразу от нескольких классов.

Стереотипы зависимости

Для большинства отношений достаточно обычной зависимости без каких-либо дополнений. Но если необходимо выделить некоторую особенность, то в UML определен целый ряд стереотипов, применимых к зависимостям. Например:

- **"bind"** - класс-клиент может использовать некоторый шаблон для своей последующей параметризации.
- **"derive"** - атрибуты класса-клиента могут быть вычислены по атрибутам класса-источника.
- **"import"** - открытые атрибуты и операции класса-источника становятся частью класса-клиента, как если бы они были объявлены непосредственно в нем.
- **"refine"** - указывает, что класс-клиент служит уточнением класса-источника в силу причин исторического характера, когда появляется дополнительная информация в ходе работы над проектом.
- **"friend"**- указывает, что классу-клиенту доступны все атрибуты и операции класса-источника не зависимо от их видимости.Этот стереотип используется для моделирования отношений, подобных отношениям между классом и его друзьями в языке C++.

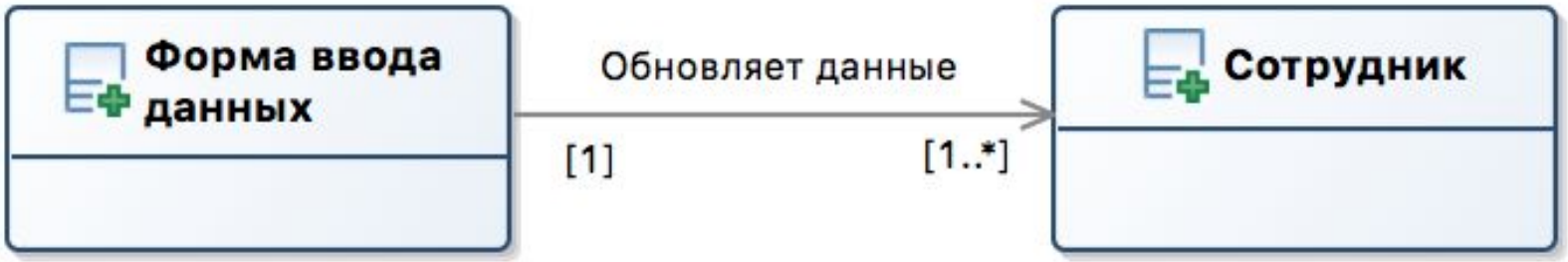
Отношение ассоциации (association)

- Отношение ассоциации соответствует наличию некоторого отношения между классами.
- Бинарная ассоциация связывает два класса.
- Множественность (multiplicity) "1" для класса “Предприятие" означает, что каждый сотрудник может работать только в одной компании. Множественность "1..*" для класса "Сотрудник" означает, что на предприятии могут работать несколько сотрудников, общее число которых заранее неизвестно и ничем не ограничено.
- Объект класса играет определенную роль в ассоциации. Предприятие выступает в роли работодателя, а сотрудник в роли работника.

Навигация

- Навигация показывает, что объекты одного класса знают о существовании объектов другого класса. Обычно объект одного класса хранит ссылку на объект другого класса.

- Навигация показывает, что объекты одного класса могут передавать сообщения объектам другого класса.

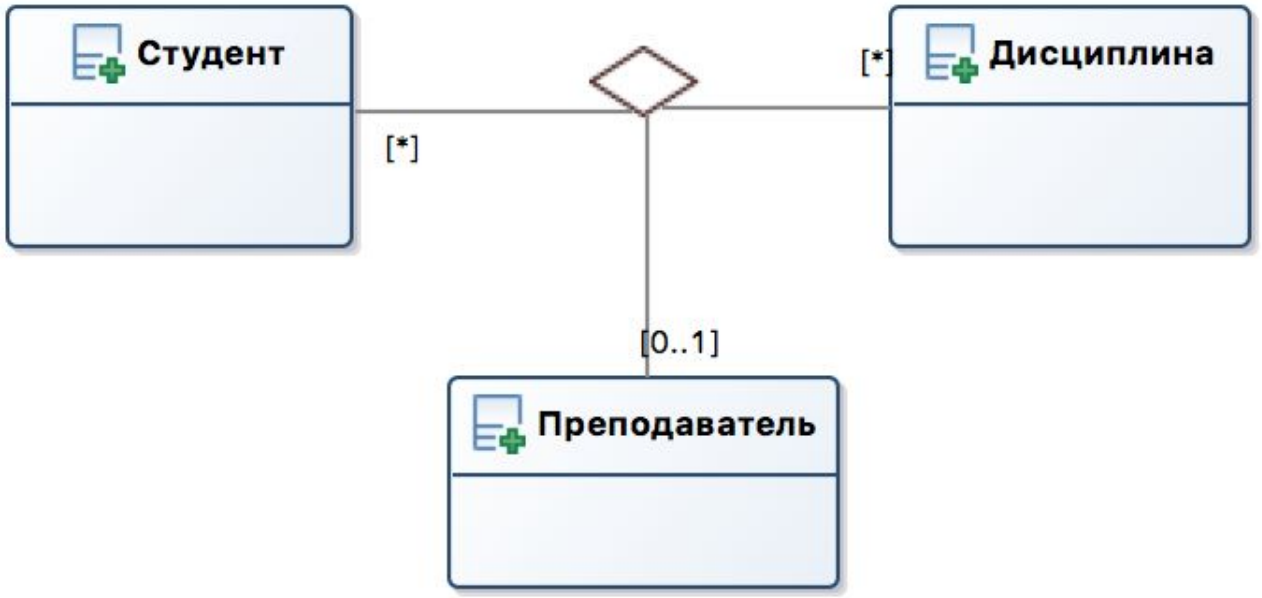


Рефлексивная ассоциация

- Рефлексивная или возвратная ассоциация - это бинарная ассоциация, связывающая класс с самим собой.
- Например, с помощью рефлексивной ассоциации можно смоделировать сеть железнодорожных станций.

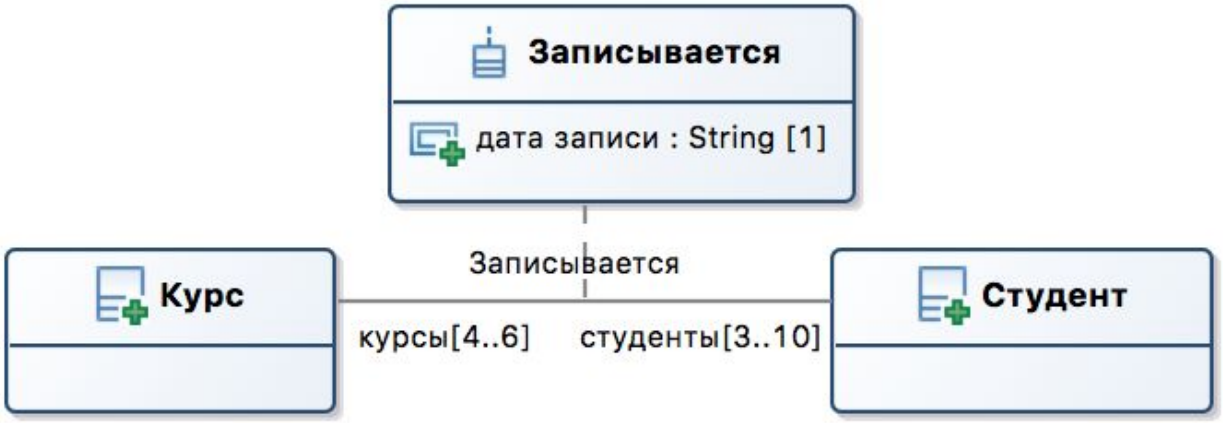
N-арная ассоциация

- В отношении ассоциации могут участвовать N классов.

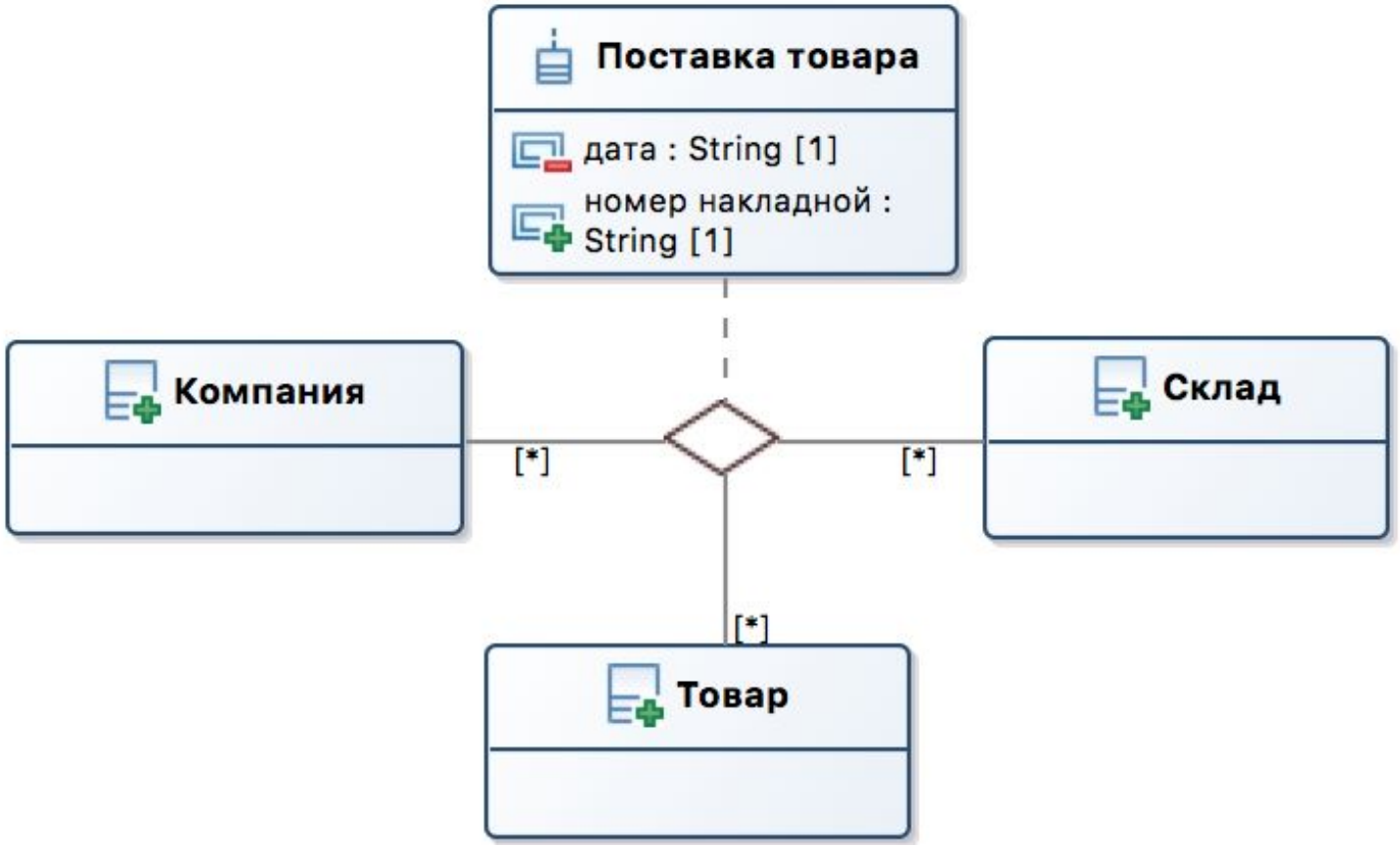


Ассоциация-класс

- Некоторый класс может быть присоединен к ассоциации пунктирной линией. Это означает, что данный класс обеспечивает поддержку свойств соответствующей N-арной (в частном случае бинарной) ассоциации, а сама N-арная ассоциация имеет атрибуты, операции и/или ассоциации.
- Такая ассоциация сама является классом с соответствующим обозначением в виде прямоугольника и является самостоятельным элементом языка UML - ассоциацией-классом (Association Class).



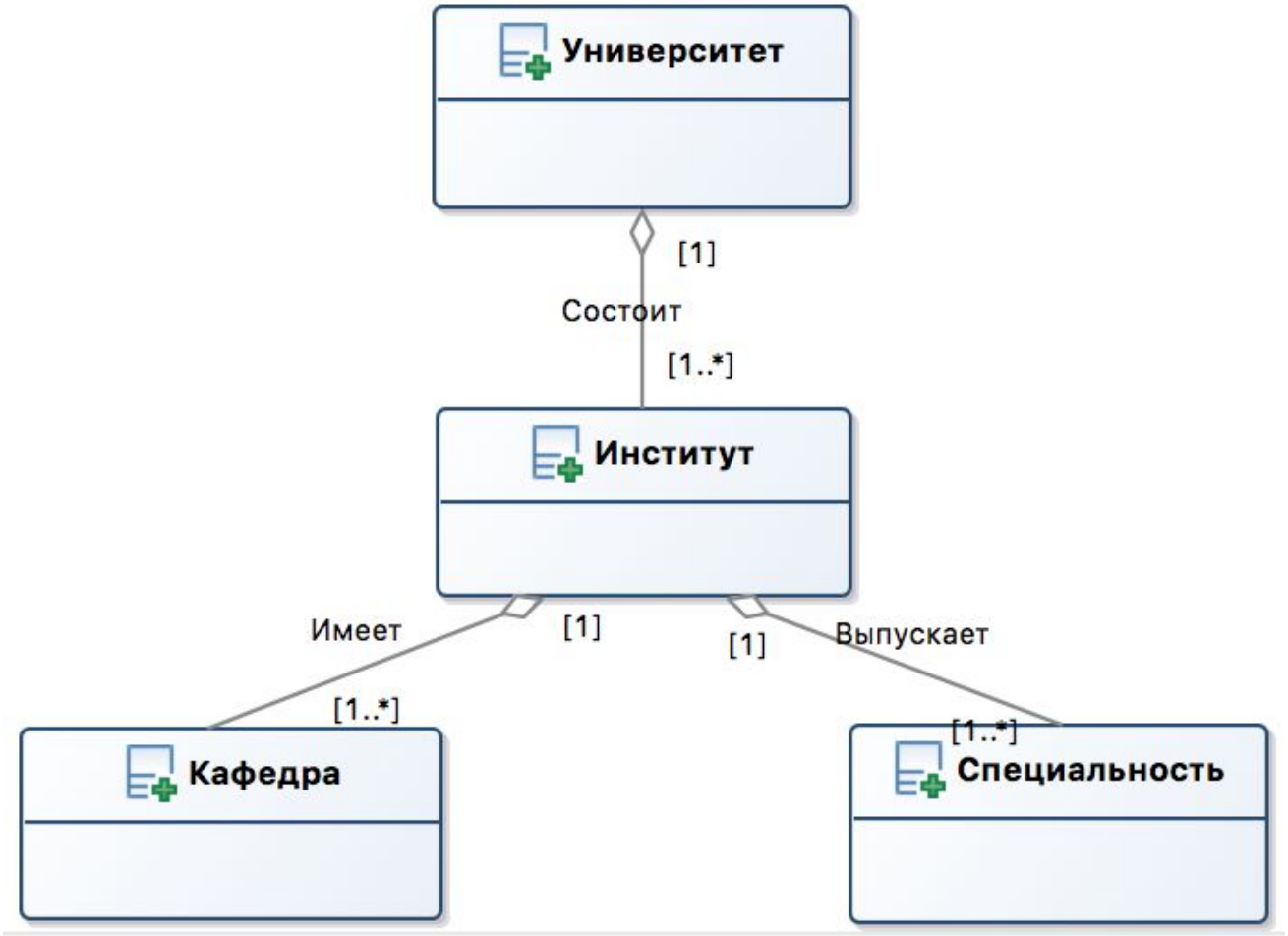
Пример ассоциации-класса для тернарной связи



Отношение агрегации (aggregation)

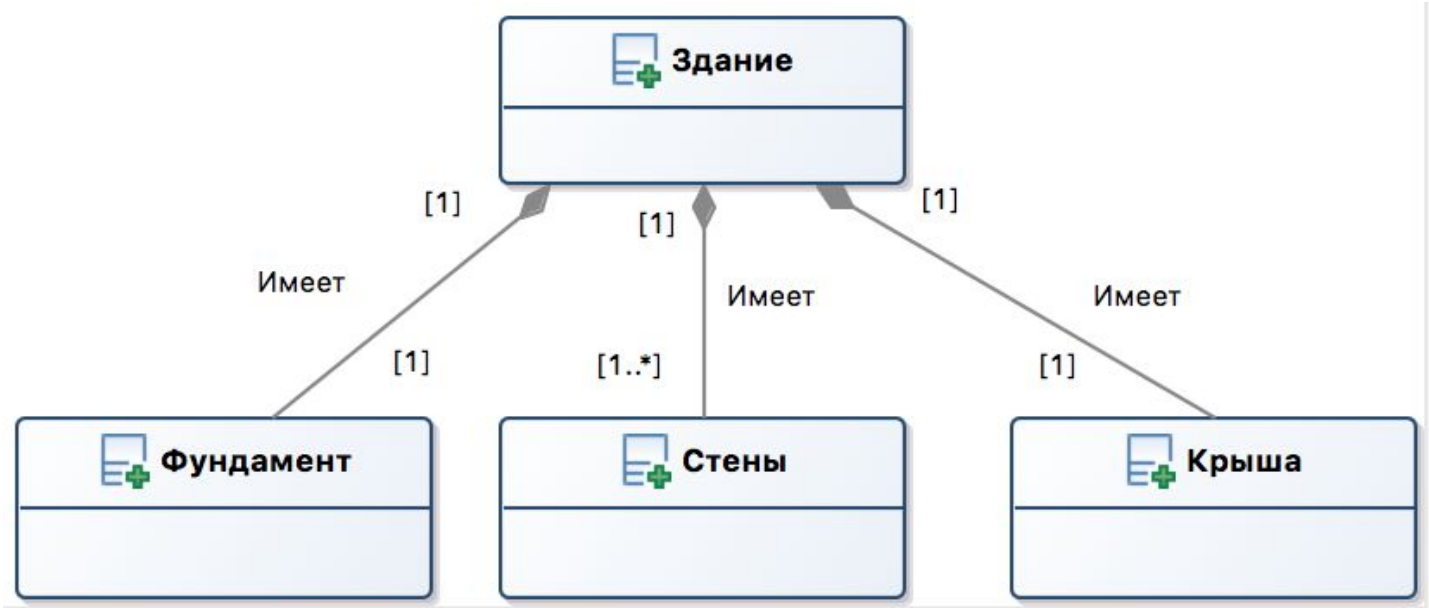
- Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности.
- Отношение агрегации показывает, из каких компонентов состоит система и как они связаны между собой.
- Это отношение позволяет описать декомпозицию или разбиение сложной системы на более простые составные части, которые также могут быть подвергнуты декомпозиции.

Пример агрегации



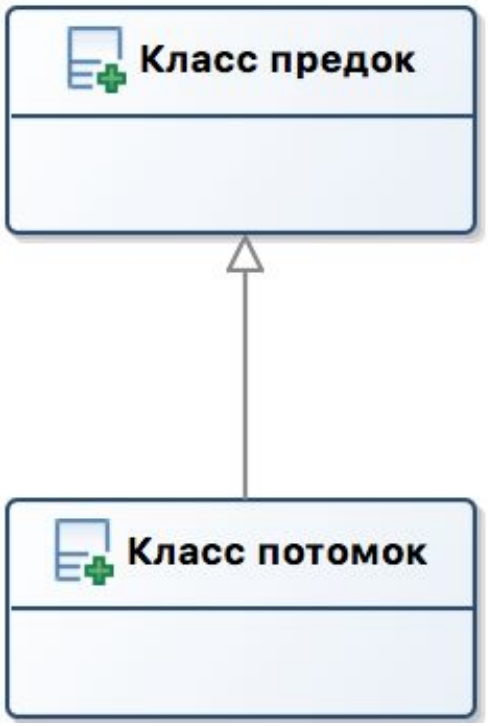
Отношение композиции (composition)

- Отношение композиции является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого.
- Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части.



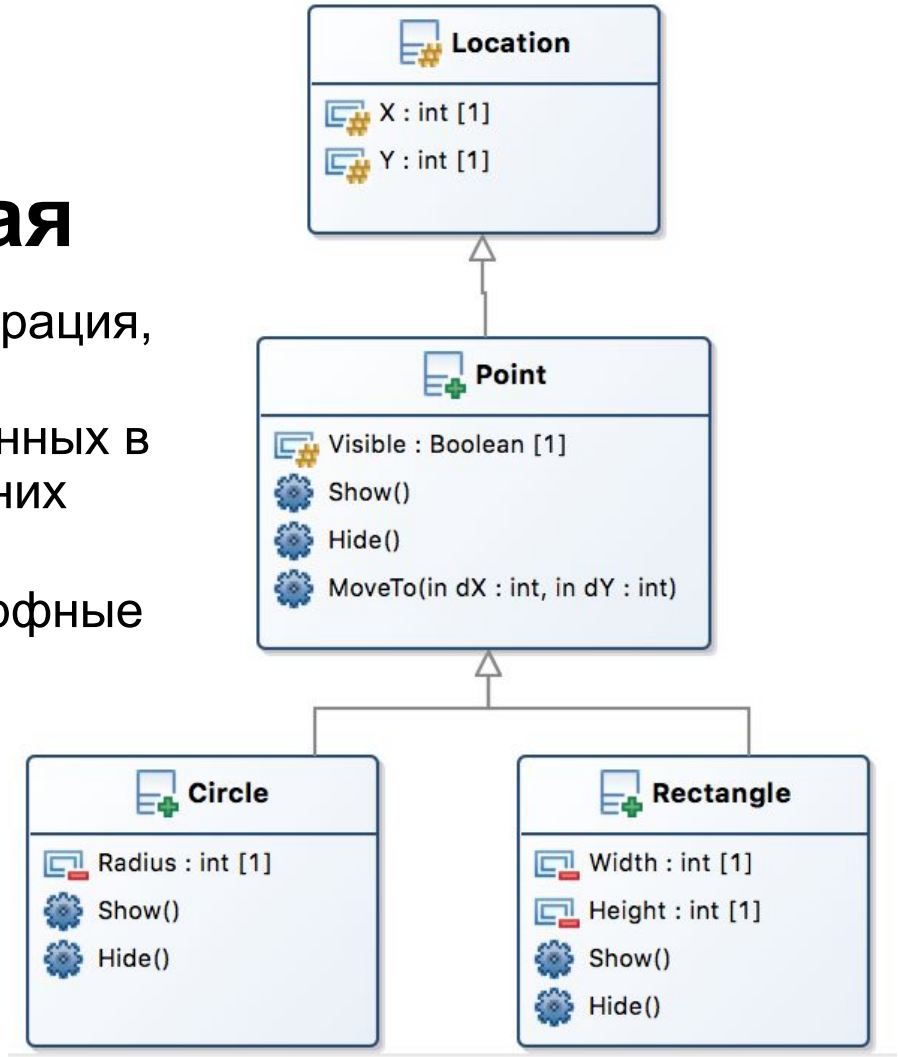
Отношение обобщения (generalization)

- Отношение обобщения является отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком).
- Отношение обобщения описывает иерархическое строение классов и наследование их свойств и поведения.
- Класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка.



Пример отношений обобщения

- **Полиморфная операция** - операция, имеющая несколько реализаций, определенных в родительском и дочерних классах.
- Show и Hide - полиморфные операции.



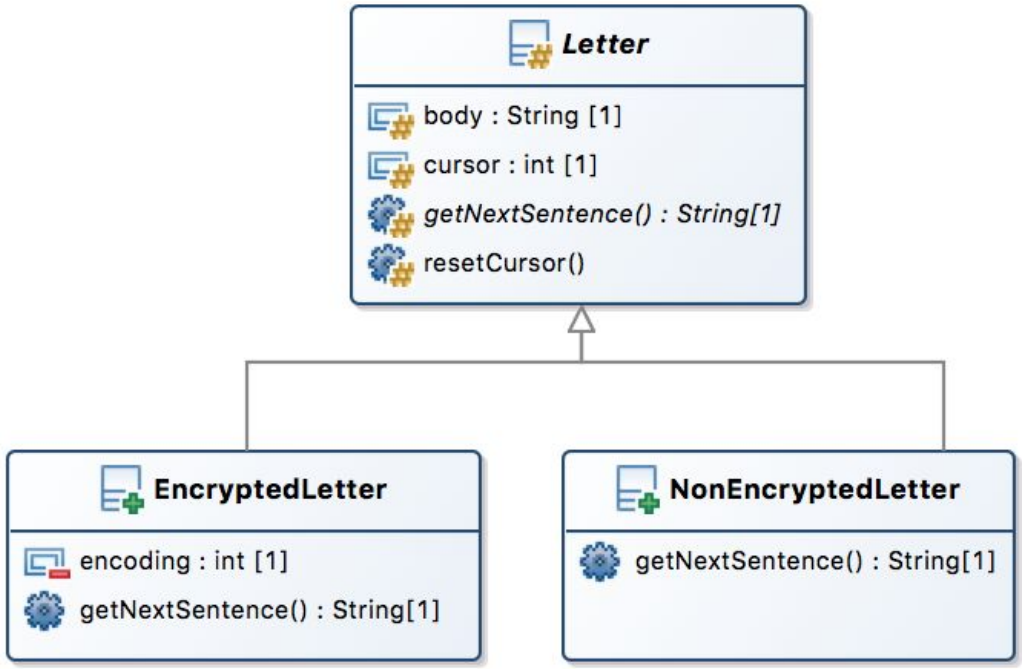
Абстрактный класс

- Абстрактный класс (abstract class) – это класс, у которого **нет экземпляров**, т.е. нельзя создать объект этого класса.
- Абстрактный класс **нуждается в уточнении**. Требуется создать потомок абстрактного класса, для которого можно создавать объекты.
- Класс, у которого **не реализована хотя бы одна операция**, автоматически является абстрактным.
- Класс, у которого **реализованы все операции** может быть абстрактным, но это нужно специально указать.
- ***Имя абстрактного класса*** пишется

Пример абстрактного класса

- **Абстрактный класс** *Letter* (письмо) содержит операции:
  - *getNextSentence* - возвращает текст следующего не прочитанного предложения (это абстрактная операция)
  - resetCursor – устанавливает курсор в начало текста
- **Конкретные классы:**
  - EncryptedLetter (зашифрованное письмо)
  - No

Соде



Интерфейс

- Интерфейс (interface) – это абстрактный класс, который имеет только абстрактные операции.
- Интерфейс служит для определения некоторой услуги, которую предлагает класс.
- Логическую реализацию интерфейса может осуществлять один или несколько классов.
- Класс, который реализует интерфейс, наследует все его операции и реализует их.
- Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов или функциональности для актеров.

Пример интерфейса

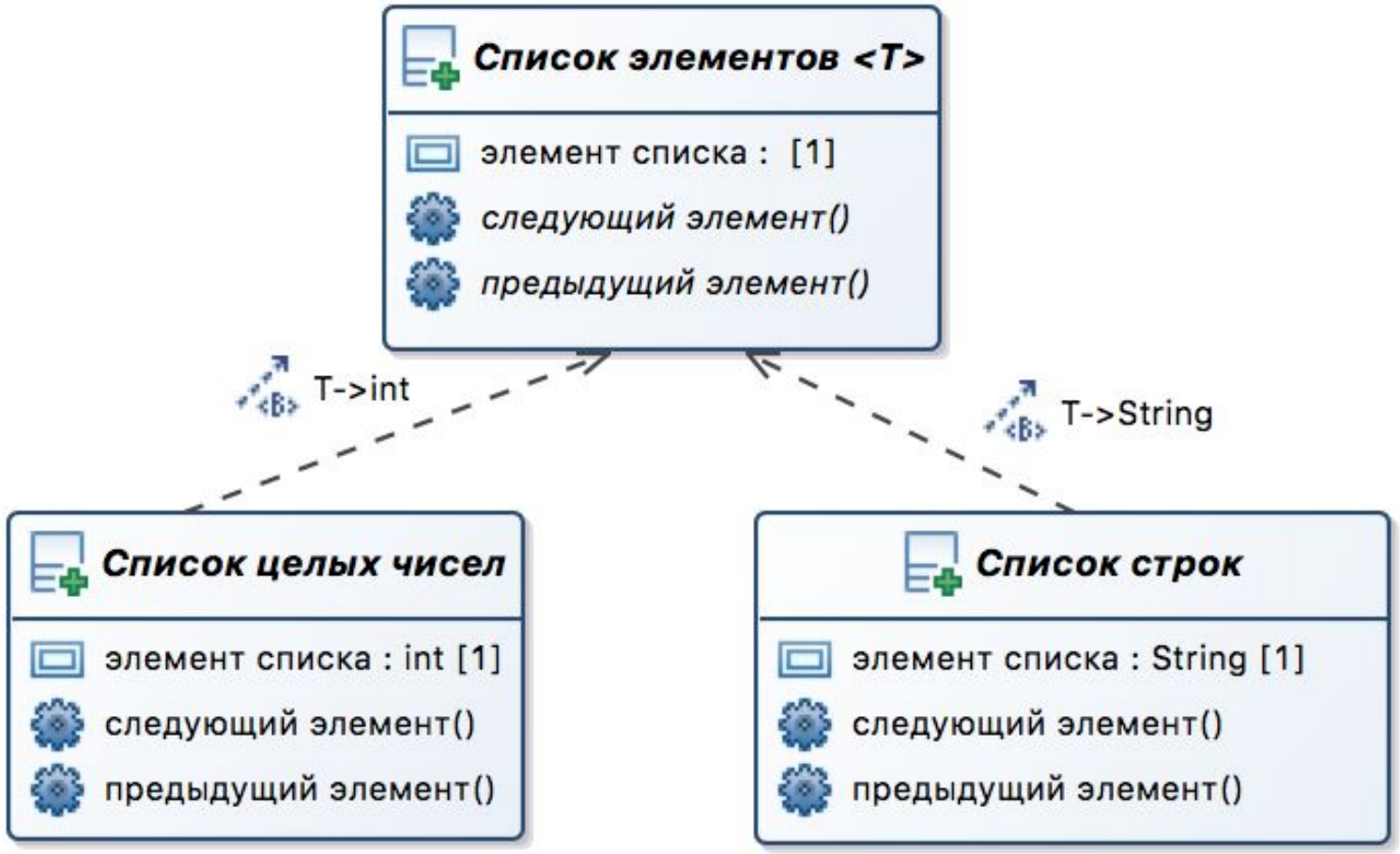
Интерфейс на диаграмме классов.

Интерфейс на диаграмме вариантов использования.

Параметризованный класс (parameterized class)

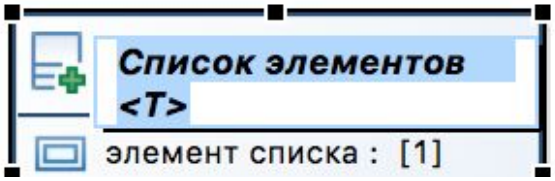
- Параметризованный класс или шаблон (template) предназначен для обозначения такого класса, который имеет один или несколько формальных параметров.
- Параметризованный класс определяет целое семейство или множество классов, каждый из которых может быть получен связыванием этих параметров с действительными значениями.
- Параметрами шаблонов могут быть классы или значения.
- Шаблон не может быть непосредственно использован в качестве класса, поскольку содержит неопределенные параметры.
- В качестве шаблона выступает некоторый суперкласс, параметры которого уточняются в его классах-потомках. Между ними существует отношение зависимости с ключевым словом "bind", когда класс-клиент может использовать некоторый шаблон для своей последующей параметризации.

Пример параметризованного класса

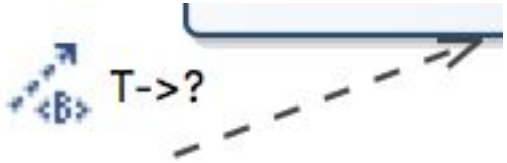


Создание параметризованного класса и зависимости "bind"

- Для создания параметризованного класса создайте обычный класс, а затем на диаграмме (не в окне редактирования имени) добавьте к имени класса параметр в угловых скобках.



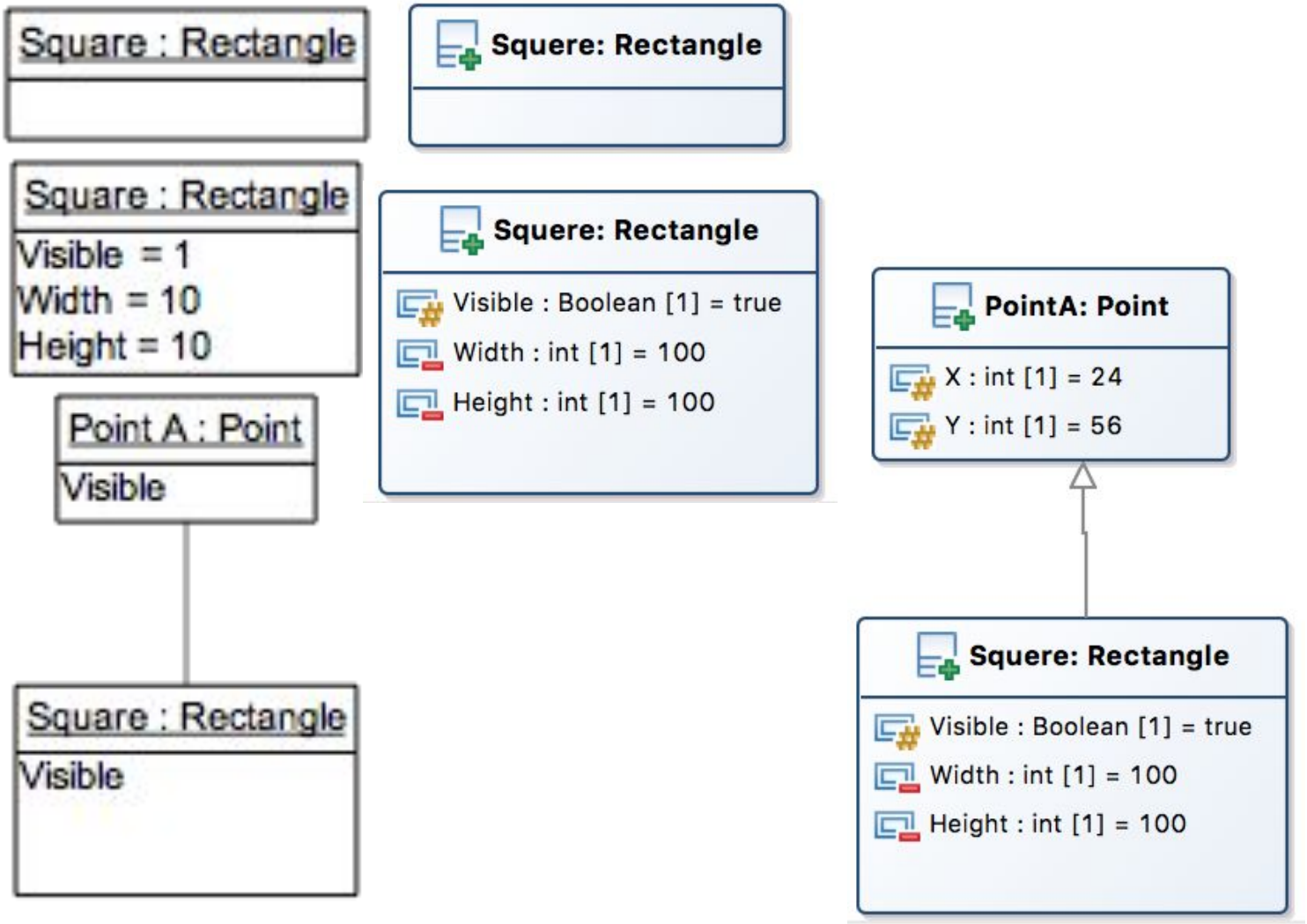
- Для создания зависимости "bind" соедините обычный класс и параметризованный класс зависимостью со стереотипом "B", а затем на диаграмме вместо знака ? запишите нужный параметр.



Объекты

- Объект (object) является отдельным экземпляром класса, который создается на этапе выполнения программы. Он имеет свое собственное имя и конкретные значения атрибутов.
- Между объектами существуют те же связи, что и между классами модели. Все связи между объектами изображаются сплошными линиями с помощью связей Link.
- Объекты также могут присутствовать на других диаграммах UML (диаграмме деятельности, диаграммах взаимодействия).
- В UML Designer диаграмма объектов пока не реализована. Объекты можно рисовать на диаграмме классов и использовать отношения, заданные для классов этих объектов.

Пример объекта “квадрат”

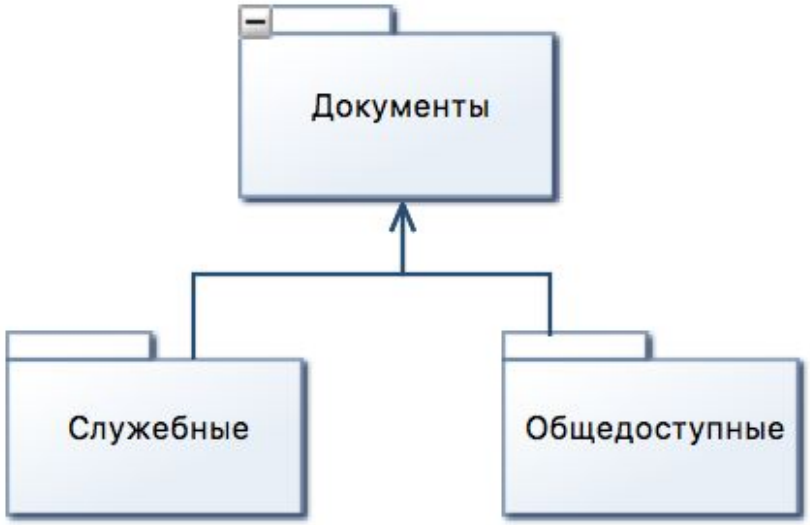


Пакеты

- Пакет (package)- это способ организации элементов модели в более крупные блоки.
- У каждого пакета должно быть имя, отличающее его от других пакетов.
- Пакет может владеть другими элементами, в том числе классами, интерфейсами, компонентами, диаграммами и даже другими пакетами.
- Каждый элемент может принадлежать только одному пакету.
- Пакеты отображают архитектуру всей системы (деление системы на подсистемы и зависимости между ними).
- Графическое изображение пакета

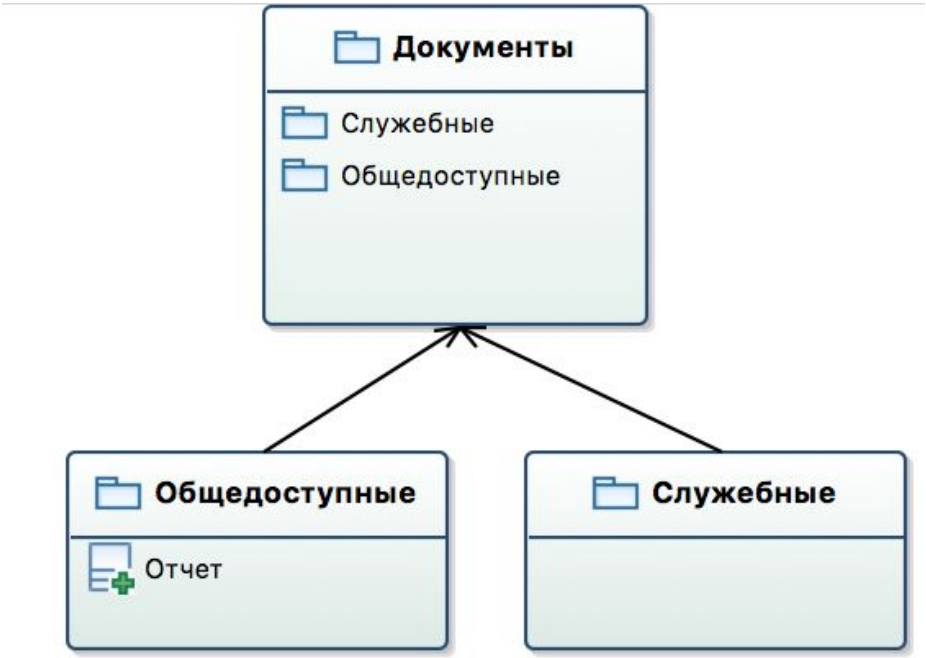
Вложенные пакеты

- В UML Designer вложенность одного пакета в другой пакет показывается на диаграмме Package Hierarchy с помощью отношения Contained in.



Вложенные пакеты и элементы

- В UML Designer вложенные пакеты и элементы (например, классы) показываются на диаграмме Package Containment.



Имена вложенных элементов

- Пакет определяет некоторое пространство имен.
- Все элементы внутри пакета должны иметь уникальные имена.
- При обращении к элементу внутри пакета достаточно указать его имя, например, класс **Отчет.**
- Если к элементу идет обращение из другого пакета, то нужно указать его полное имя, состоящее из одного имени или последовательности имен пакетов (если пакеты вложены один в другой) и имени элемента.
- Например: **Документы:: Общедоступные::Отчет**

Видимость элементов пакета

- Видимость определяет доступность элементов в пределах пакета и из других пакетов. Возможные значения:

“+” – **общедоступный** элемент (например, класс) виден как внутри пакета, так и из других пакетов, использующих данный пакет. Говорят, что эти элементы экспортируются пакетом.

“-” – **закрытый** элемент пакета недоступен из других пакетов и моет использоваться только элементами данного пакета.

Зависимости между пакетами

- Между пакетами может быть установлено отношение зависимости (dependency).
- Зависимость показывает, что элементы одного пакета могут использовать элементы другого пакета.
- Пакет B - поставщик зависимости. Пакет A - клиент зависимости.
- Элементы одного пакета не видят элементов другого пакета, если между пакетами не установлено отношение зависимости.

Стереотипы зависимостей

- **usage** - пакет клиент использует открытый элемент пакета поставщика. Для доступа указывается имя пакета и имя элемента.
- **import** - открытые элементы поставщика добавляются к открытым элементам клиента, т.е. клиент может работать с открытыми элементами поставщика, но не наоборот. Для обращения к ним из клиента достаточно указать только имя.
- **access** - открытые элементы поставщика B добавляются к закрытым элементам клиента A. Если третий пакет C будет импортировать клиента A, то он уже не увидит открытые элементы поставщика B.
- **merge** - открытые элементы поставщика объединяются с элементами клиентского пакета для создания новых расширенных клиентских элементов. Например, к атрибутам и операциям класса A клиента добавляются атрибуты и операции класса A поставщика.

- **trace** - зависимость между историческими реализация одного и