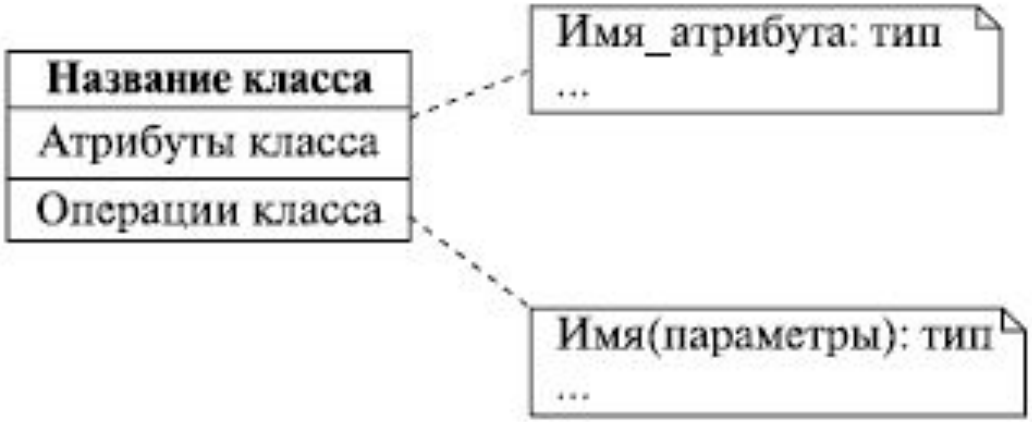
ДИАГРАММА КЛАССОВ

Изображение класса

- *Класс* на диаграмме изображается в виде прямоугольника, разделенного горизонтальными линиями на три части.
- В первой части указывается название класса.
- Вторая часть содержит перечень атрибутов класса, которые характеризуют тот или иной *объект* этого класса в модели *предметной области*.
- Третья часть содержит перечень операций, отражающих его поведение в модели *предметной области*



Модификаторы видимости

- ✔ *В программировании инкапсуляция обеспечивается с помощью модификаторов видимости*
- ✔ *Модификаторы видимости* ограничивают *доступ* к атрибутам

Модификаторами доступа обозначаются специальными символами слева от их имен:

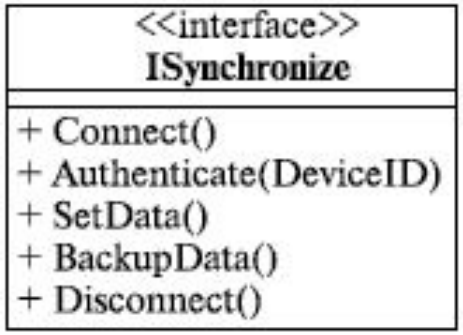
Символ	Значение
+	`public` - открытый доступ
–	`private` - только из операций того же класса
#	`protected` - только из операций этого же класса и классов, создаваемых на его основе

Интерфейс

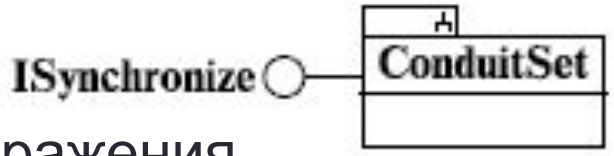
- Интерфейс - это логическая *группа* открытых (public) операций объекта.
- Один и тот же *объект* может иметь несколько интерфейсов.
- *Интерфейс* отражает внешние проявления объекта, показывает, каким образом осуществляется взаимодействие с ним, скрывая остальные детали, не имеющие отношения к процессу взаимодействия.
- *Интерфейс* всегда реализуется некоторым классом, который в таком случае называют *классом, поддерживающим интерфейс*. Как мы уже говорили ранее, один и тот же *объект* может иметь несколько интерфейсов. Это означает, что *класс* этого объекта реализует все *операции* этих интерфейсов.

Изображение интерфейсов

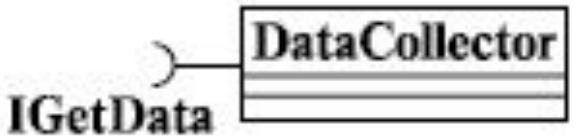
- *Первый способ - Класс* со стереотипом <<interface>>, если нужно показать, какие именно *операции* предоставляет *интерфейс*



- *Второй способ* - В виде кружочка или, как говорят, «леденца», если перечень операций не важен



- *Третий способ* используется для изображения интерфейсов, **требующихся** объекту для выполнения его работы

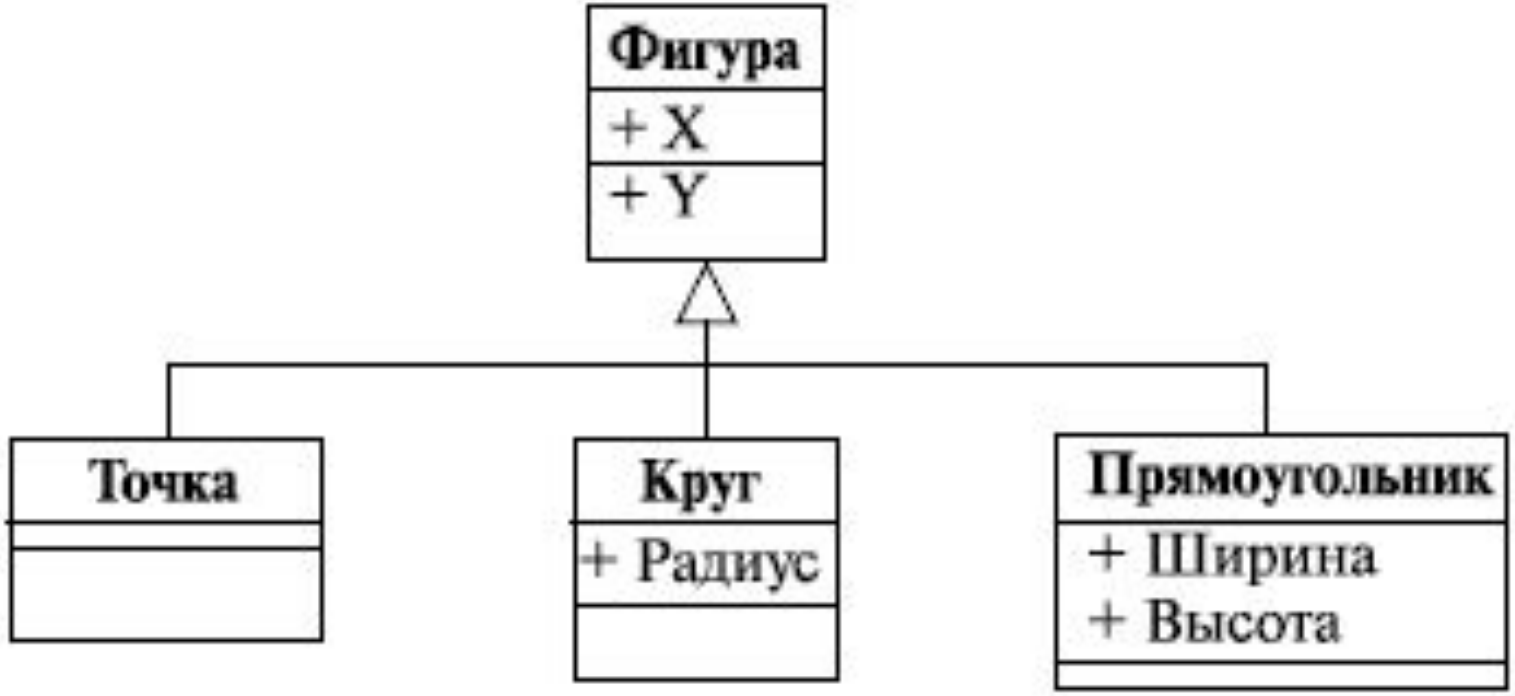


Почему стоит использовать уже существующие классы:

- Во-первых, идя этим путем, мы пользуемся плодами ранее принятых решений. Действительно, если когда-то мы уже решили некоторую проблему, зачем начинать все "с нуля", повторяя уже однажды проделанные действия?
- Во-вторых, **таким образом мы делаем решение мобильным и расширяемым**. Используя уже существующие классы и создавая на их основе новые, мы можем развивать решение практически неограниченно, добавляя лишь необходимые нам в данный момент детали - атрибуты и *операции*.
- В-третьих, существующие классы, как правило, **хорошо отлажены и показали себя в работе**. Разработчику не надо тратить время на *кодирование*, отладку, тестирование и т. д., - мы работаем с хорошо отлаженным и проверенным временем кодом, который зарекомендовал себя в других проектах и в котором уже выявлено и исправлено большинство ошибок.

Обобщение

- **Обобщение** - это *отношение* между более общей сущностью, называемой суперклассом, и ее конкретным воплощением, называемым подклассом.
- При этом все атрибуты и операции суперкласса независимо от модификаторов видимости **входят в состав подкласса.**
- *Обобщение* (или, как часто говорят, *наследование*) на диаграммах обозначается незакрашенной треугольной стрелкой, направленной на *суперкласс*

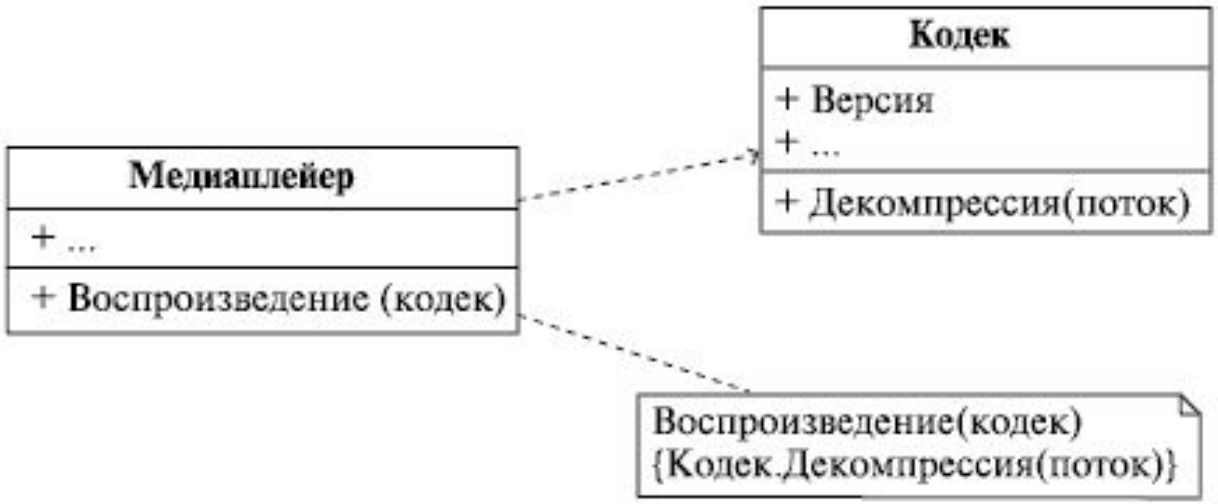

Фигура
+ X
+ Y
Точка
Круг
+ Радиус
Прямоугольник
+ Ширина
+ Высота

Полиморфизм

- Так же как классы-потомки наследуют атрибуты и *операции* суперкласса, они могут наследовать и их интерфейсы - то есть объекты абсолютно разной природы могут иметь один и тот же *интерфейс*.
- Работа механизма полиморфизма основана на совпадении сигнатуры метода, объявленного в интерфейсе, и сигнатуры самого метода.
- Методы внутри классов-потомков могут быть переопределены, их реализации будут различными, а сигнатуры останутся неизменными. Таким образом, выполняя одни и те же *операции*, разные объекты могут вести себя по-разному.

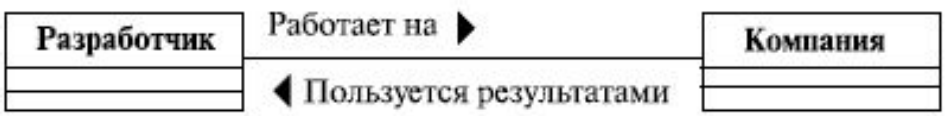
Отношения между классами

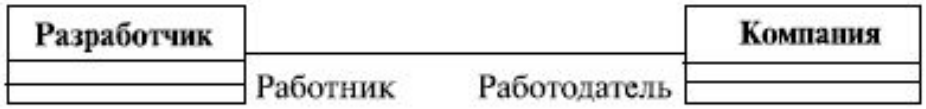
- **Зависимость** возникает тогда, когда реализация класса одного объекта зависит от спецификации операций класса другого объекта. И если изменится спецификация операций этого класса, нам неминуемо придется вносить изменения и в зависимый *класс*.

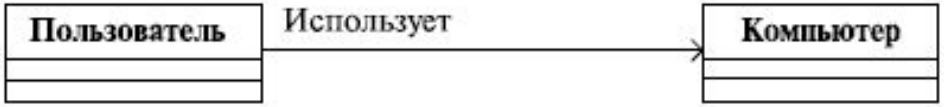


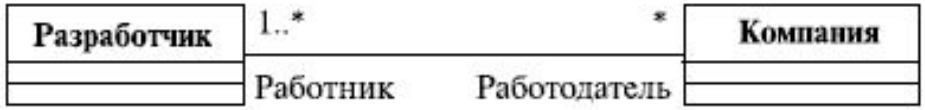
Отношения между классами

- **Ассоциация**. Это просто *связь* между объектами, по которой можно между ними перемещаться.
- *Ассоциация* может иметь имя, показывающее природу отношений между объектами, при этом в имени может указываться направление чтения связи при помощи треугольного маркера.
- Однонаправленная *ассоциация* может изображаться стрелкой



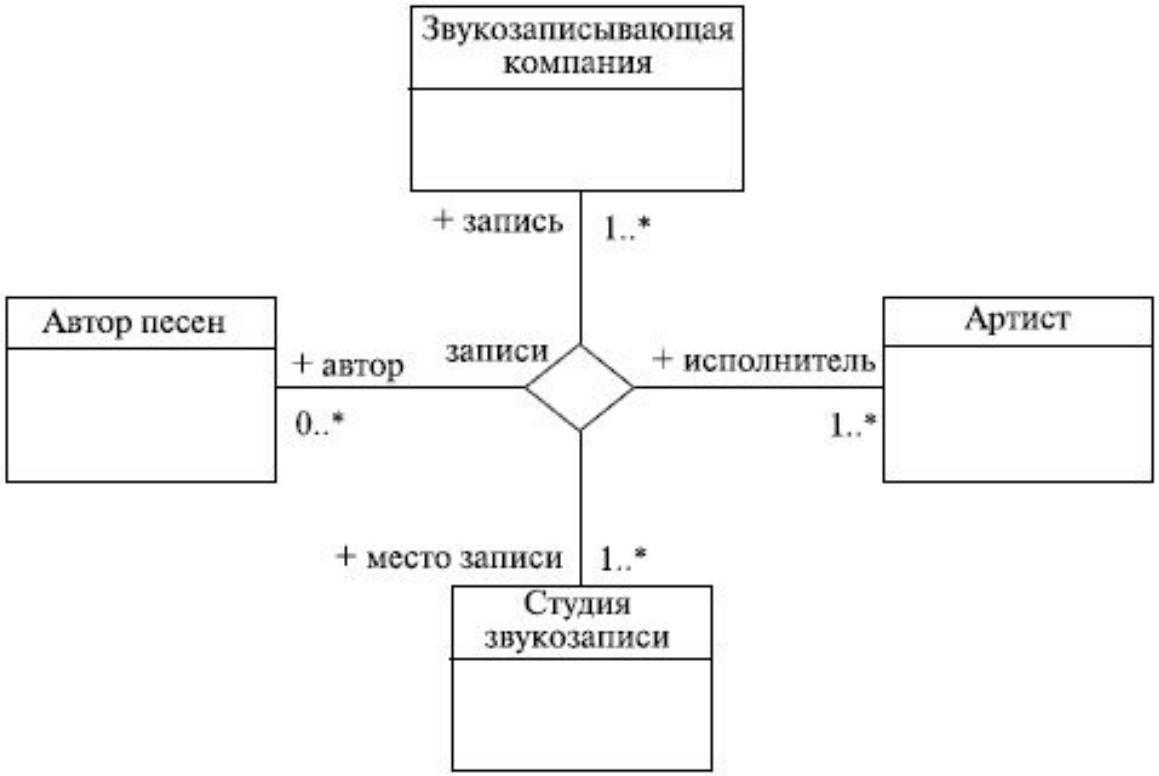






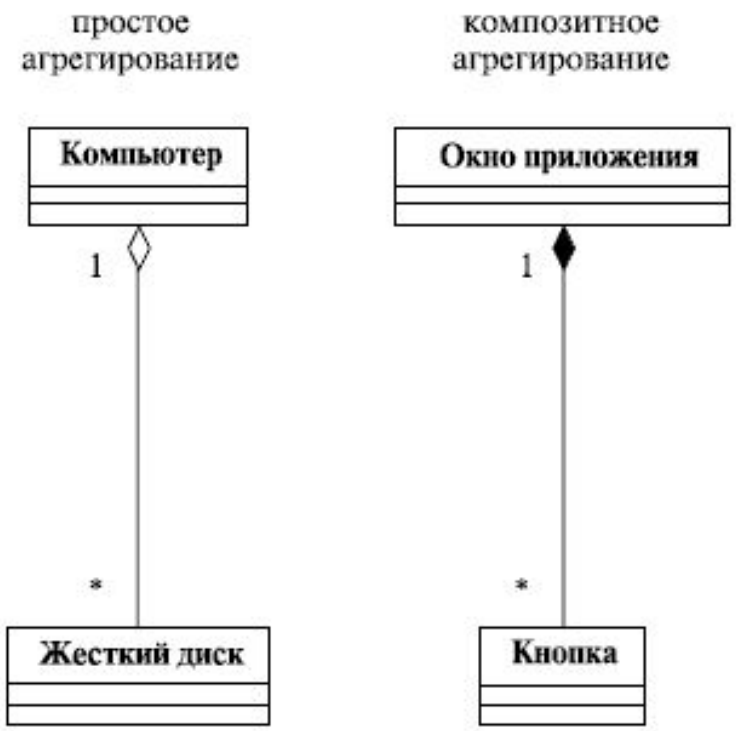
Типы ассоциаций

- *Ассоциация* может объединять три и более класса. В этом случае она называется **n-арной** и изображается ромбом на пересечении линий



Типы ассоциаций

- Ассоциации могут подразумевать и более сложное *отношение* между классами, например, *связь* типа "часть-целое". Такой вид ассоциации называется **ассоциацией с агрегированием**.
- В этом случае один *класс* имеет более высокий статус (целое) и состоит из низших по статусу классов (частей). При этом выделяют простое и композитное *агрегирование* и говорят о собственно **агрегации** и **композиции**.
- **Простая *агрегация*** предполагает, что части, отделенные от целого, могут продолжать свое существование независимо от него.
- **Под композитным же агрегированием** понимается ситуация, когда целое владеет своими частями и их время жизни соответствует времени жизни целого, т. е. независимо от целого части существовать не могут.



Типы ассоциаций

- В отношении между двумя классами сама *ассоциация* тоже может иметь свойства и, следовательно, тоже может быть представлена в виде класса.

Выводы

- Инкапсуляция защищает внутреннее устройство объекта и реализуется путем ограничения доступа к атрибутам и операциям класса из других частей программы.
- Обобщение позволяет повторно использовать уже существующие решения, создавая новые классы путем наследования от имеющихся классов.
- Полиморфизм позволяет работать с группой разнородных объектов одинаковым образом, не задумываясь о различиях в реализации.