**Аспектно-ориентированное программирование (АОП)** — это парадигма программирования являющейся дальнейшим развитием процедурного и объектно-ориентированного программирования (ООП). Идея АОП заключается в выделении так называемой сквозной функциональности.

Для этого к уже существующему коду добавляется дополнительное поведение, без изменений в изначальном коде. Иными словами, мы как бы навешиваем сверху на методы и классы дополнительную функциональность, не внося поправки в модифицируемый код.

## Применение АОП

Аспектно-ориентированное программирование предназначено для решения сквозных задач, которые могут представлять собой любой код, многократно повторяющийся разными методами, который нельзя полностью структурировать в отдельный модуль.

В качестве еще одного примера можно привести логирование.

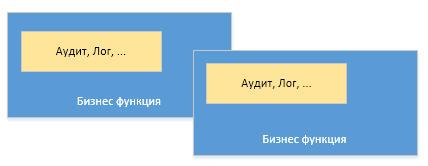
У использования АОП подхода к логированию есть несколько преимуществ по сравнению с ручной вставкой логирования:

1. Код для логирования легко внедрять и удалять: всего-то нужно добавить или удалить пару конфигураций некоторого аспекта.
2. Весь исходный код для логирования хранится в одном месте и не нужно находить вручную все места использования.
3. Код, предназначенный для логирования, можно добавить в любое место, будь то уже написанные методы и классы или же новый функционал. Это уменьшает количество ошибок разработчика.  
   Также при удалении аспекта из конфигурации конструкции можно быть абсолютно уверенным, что весь код трассировки удален и ничего не пропущено.
4. Аспекты — это вынесенный отдельно код, который можно многократно переиспользовать и улучшать.

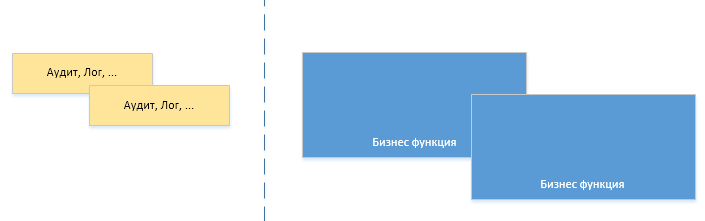
Также АОП используется для обработки исключений, кеширования, выноса некоторого функционала, чтобы сделать его переиспользуемым.

### **Выделение сквозной функциональности**

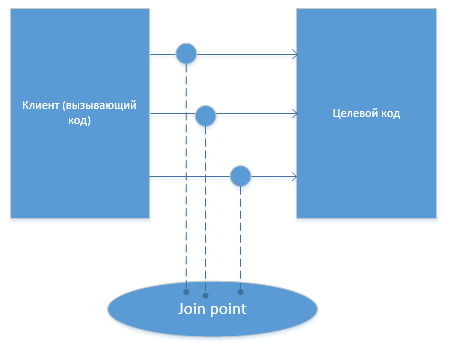
До



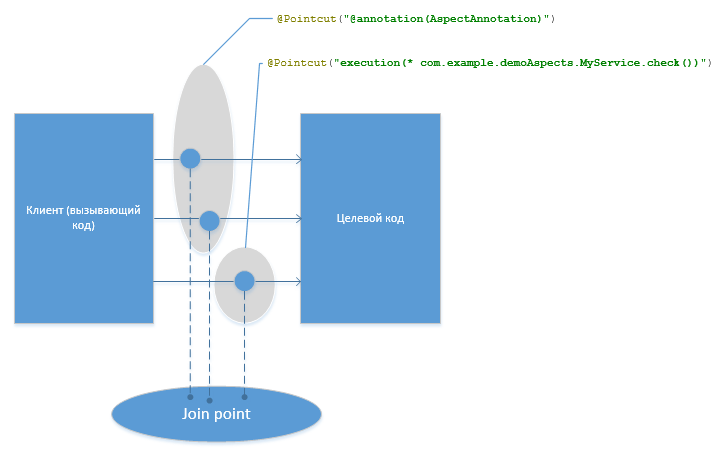
и после

Т.е. есть функциональность которая затрагивает несколько модулей, но она не имеет прямого отношения к бизнес коду, и ее хорошо бы вынести в отдельное место, это и показано на рисунке выше.

**Точка соединения (JOIN POINT)**

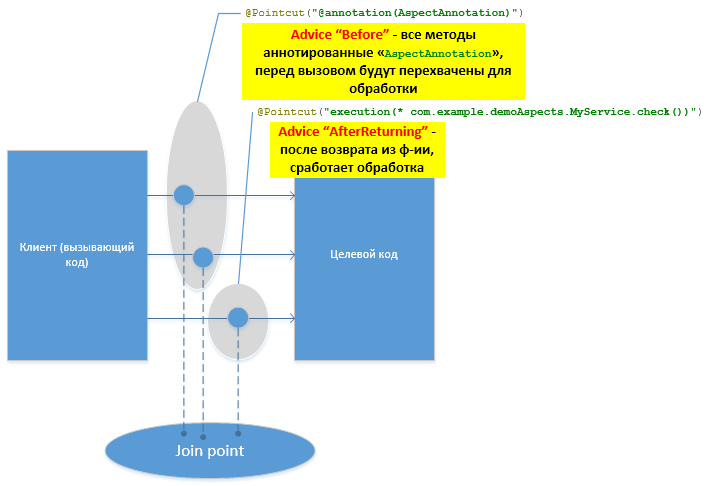
Точка в выполняемой программе (вызов метода, создание объекта, обращение к переменной), где следует применить совет. Иначе говоря, это некоторое регулярное выражение, с помощью которого и находятся места для внедрения кода (места для применения советов).

**Срез (POINTCUT)**

Набор **точек соединения**. Срез определяет, подходит ли данная точка соединения к данному совету.

Это может быть одна и более точек. Правила запросов точек очень разнообразные, на рисунке выше, запрос по аннотации на методе и конкретный метод. Правила можно объединять по &&, ||.

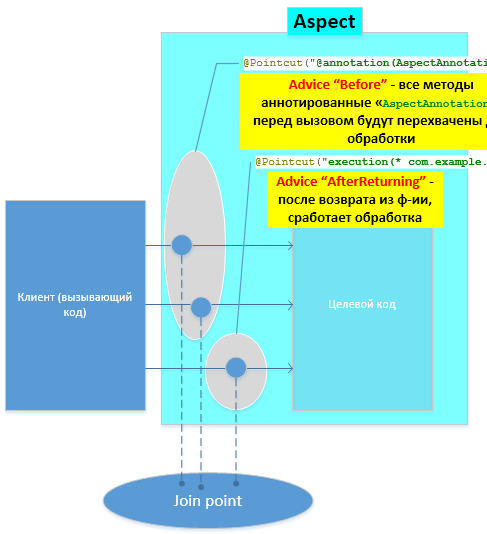
**Совет (ADVICE)** — это дополнительная логика, код, который вызывается из точки соединения.   
Совет может быть выполнен до, после или вместо точки соединения (о них — ниже).



Возможные виды советов:

1. **Перед (Before)** — советы данного типа запускаются перед выполнением целевых методов — точек соединения. При использовании аспектов в виде классов мы берем@Before аннотацию, чтобы пометить тип совета как идущий перед.
2. **После (After)** — советы, которые выполняются после завершения выполнения методов — точек соединения, как в обычных случаях, так и при бросании исключения.  
   При использовании аспектов в виде классов мы можем использовать @After аннотацию для указания, что это совет, идущий после.
3. **После возврата (After Returning)** — данные советы выполняются только в том случае, когда целевой метод отрабатывает нормально, без ошибок.  
   Когда аспекты представлены в виде классов, мы можем использовать аннотацию @AfterReturning, чтобы пометить совет как выполняемый после успешного завершения.
4. **После бросания (After Throwing)** — данный вид советов предназначен для тех случаев, когда метод, то есть точка соединения выдает исключение. Мы можем использовать этот совет для некой обработки неудачного выполнения (к примеру, для отката всей транзакции или логирования с необходимым уровнем трассировки).  
   Для аспектов-классов аннотация @AfterThrowing используется, чтобы указать, что этот совет используется при после броска исключения.
5. **Вокруг (Around)** — пожалуй, один из самых важных видов советов, который окружает метод, то есть — точку соединения, с помощью которого мы можем, к примеру, выбрать, выполнять данный метод точки соединения или нет.  
   Можно написать код совета, который будет выполняться до и после выполнения метода точки соединения.  
   В обязанности around advice входит вызов метода точки соединения и возвращение значений, если метод что-то возвращает. То есть в этом совете можно попросту сымитировать работу целевого метода, не вызывая его, и в качестве результата вернуть что-то свое.  
   При аспектах в виде классов используем @Around аннотацию для создания советов, оборачивающих точку соединения.

**Аспект (ASPECT)**



Модуль или класс, реализующий сквозную функциональность. Аспект изменяет поведение остального кода, применяя совет в точках соединения, определенных некоторым срезом. Иными словами, это комбинация советов и точек соединения.

Модуль в котором собраны описания Pointcut и Advice.

**Плетение (weaving)** — это процесс связывания аспектов с другими объектами для создания рекомендуемых прокси-объектов. Это можно сделать во время компиляции, загрузки или во время выполнения.

Есть три вида плетения:  
плетение во время компиляции  
посткомпиляционное плетение (бинарное плетение)  
плетение во время загрузки