**Necesitatea**

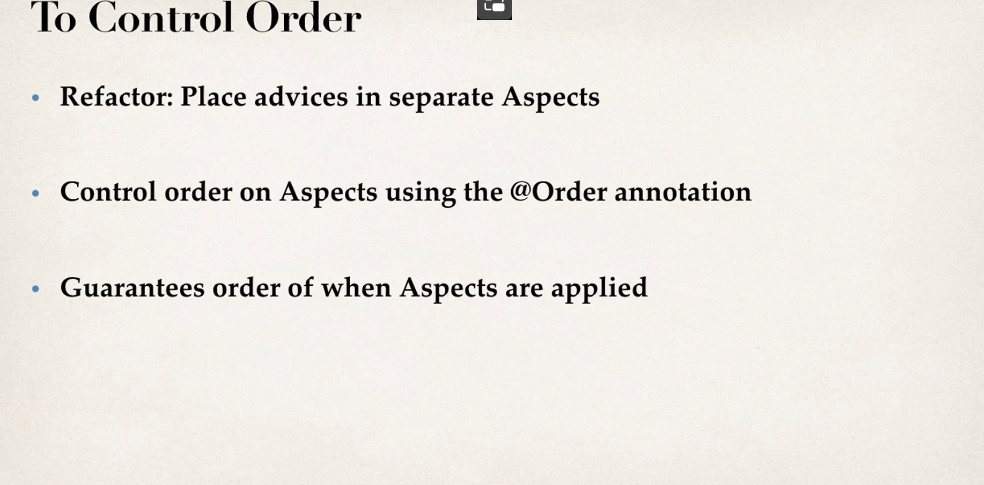
* Daca o metoda corespunde cu PointCut expression de la mai multe advices, apare problema in ce ordine se vor executa acestea. De ex:



* In asa caz, ordinea executiei poate sa fie total neclara, ceva gen random.

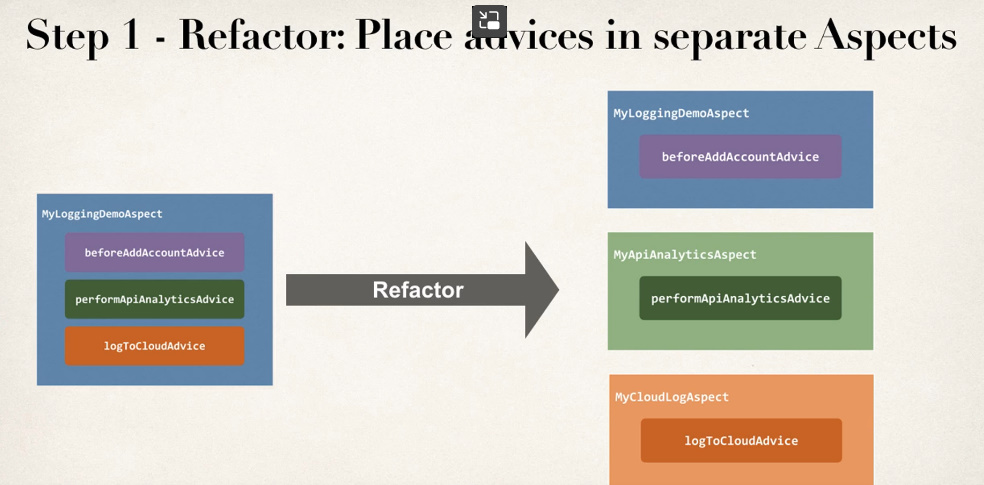
**@Order**

* Daca totusi nu vrem ca advices sa fie executate random,in orice situatie, trebuie sa punem fiecare advice intr-un aspect separat si sa folosim @Order pentru aspecte.

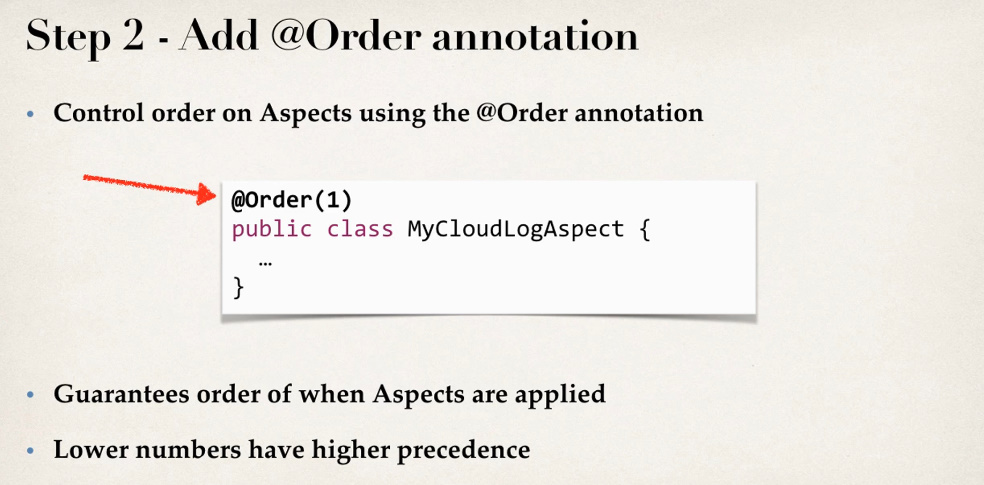


Pasi:

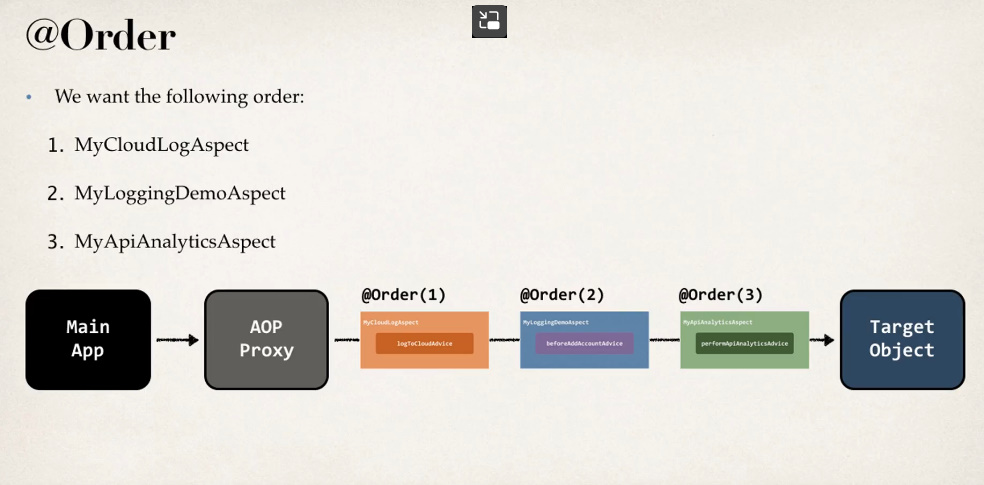
1. Separam advices in aspecte, adica in clase diferite:



1. Punem anotatia @Order(numar) la fiecare aspect:

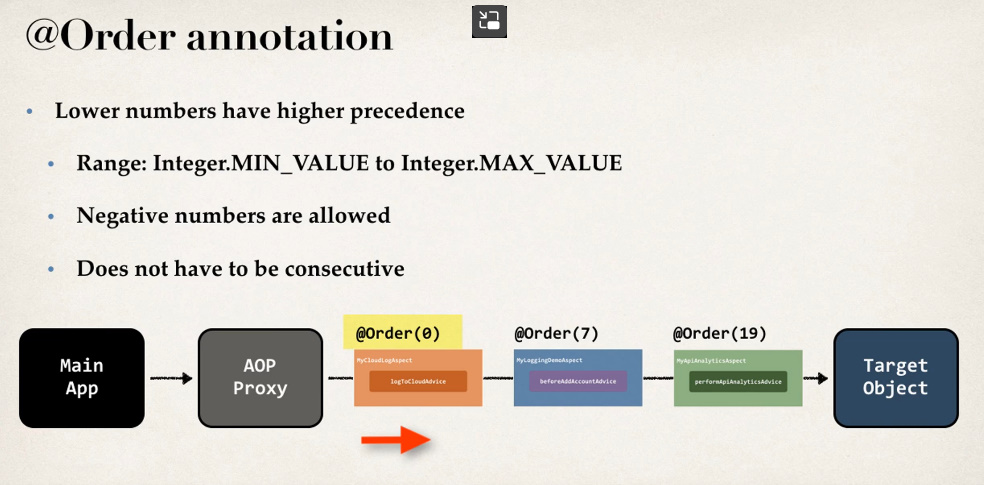


numarul dintre () este al catalea aspect rulat va fi, deci 1 e primul, 2 al doilea si tot asa

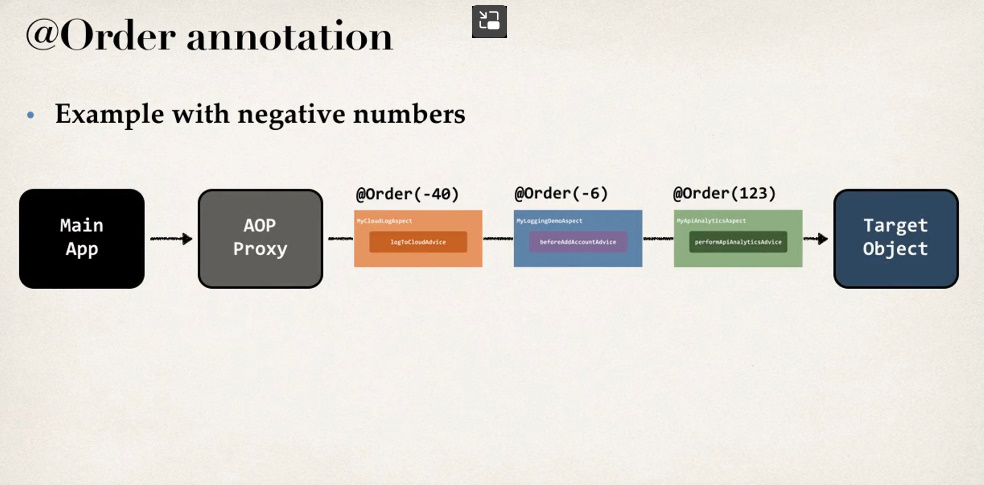


* Reguli:
* ordinea poate fi doar un numar Integer
* ordinea nu trebuie neaparat sa fie consecutiva, adica nu neaparat 1 2 3 ...

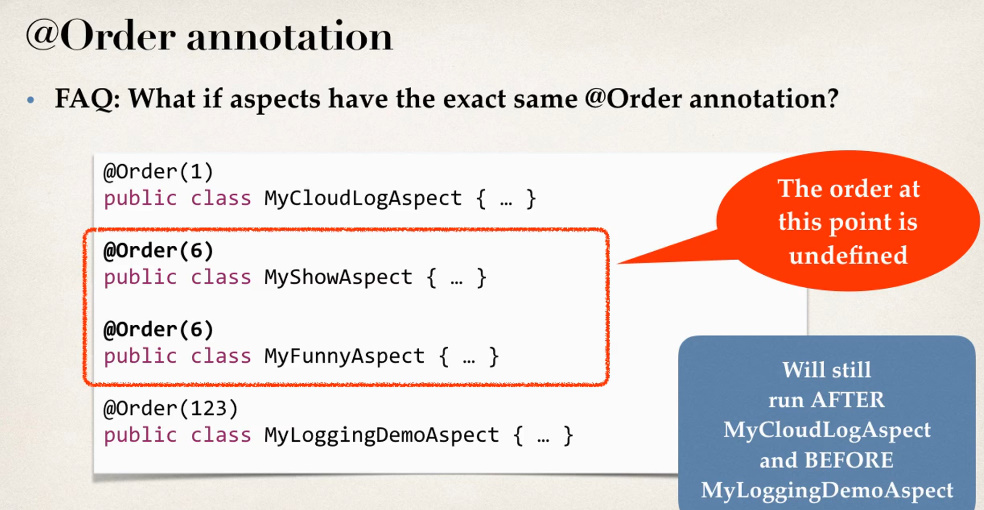
Poate fi si neconsecutiva ca 5 9 11 15, oricum Spring are grija sa execute aspects dupa valoarea numarului din @Order, incepand cu cele mai mici si apoi mergand la cele mai mari.



* Ordinea poate fi si un numar negativ. Nu conteaza ce fel de numar e, Spring ia de la cel mai mic la cel mai mare.



* Daca 2 sau mai multe aspects au aceeasi valoare la @Order(), atunci, cand va veni randul ordinii lor, iar nu se stie cum s-ar putea executa



**Problema**

Daca separam toate aceste advices in clase aspects separate, apare problema cum sa folosim la toate aceleasi @PointCut declarations fara sa le repetam in fiecare clasa.

* Solutia este de a crea o utility class unde sa stocam toate PointCut declaration si apoi aspectele class sa o mosteneasca.
* Nu uitam sa modificam private in public, de altfel nu vor putea fi accesate din clasa copil.
* Atentie! Clasa parinte nu este un bean, si nici un @Aspect.
* Clasa parinte trebuie sa fie @Aspect doar daca e abstracta! @PointCut nu trebuie sa aiba metode abstracte.