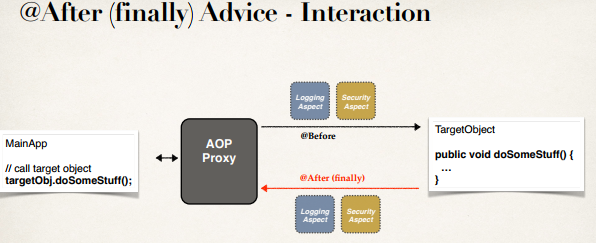
**43. AOP @After Advice Type**

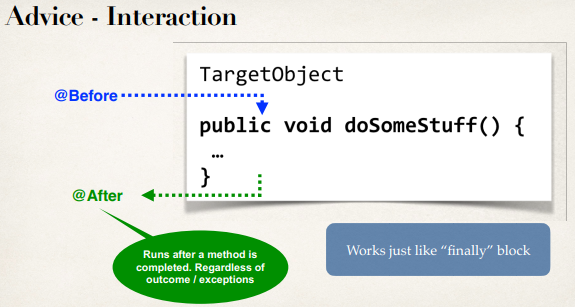
**Advice Types**:

1. **Before advice**: run before the method
2. **After returning advice**: run after the method (success execution)
3. **After throwing advice**: run after method (if exception thrown)
4. **After finally advice**: run after the method (finally)
5. **Around advice**: run before and after method

**@After (finally) Advice – Interaction**:



**Advice – Interaction**:

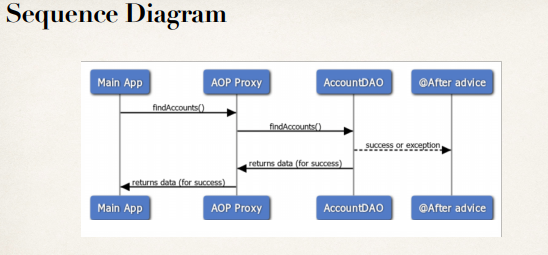


**Note**:

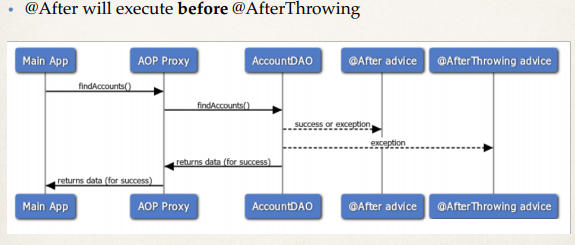
The important thing is that this **(@After)** runs after the method is completed, regardless of the outcome for success or failure, this code is always run.

This works just like "finally" block in normal try-catch-finally code in Java language.

**Sequence Diagram**:



**@After will execute before @AfterThrowing**:

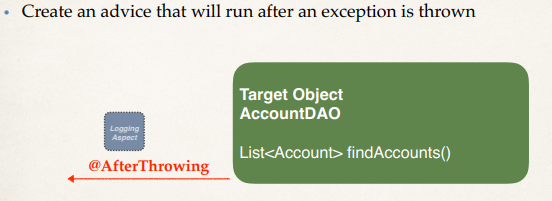


**@AfterThrowing Advice - Use Cases**:

1. Log the exception and / or perform auditing
2. Code to run regardless of method outcome
3. Encapsulate this functionality in AOP aspect for easy reuse

**Example**:

Create an advice that will run after an exception is thrown



**@After Advice**:

This advice will run after the method (finally … success/failure)

// add a new advice for @After

@After("execution(\* com.ruhul.aopdemo.dao.AccountDAO.findAccounts(..))")

**public** **void** afterFinallyFindAccountsAdvice() {

System.***out***.println("Executing @After (Finally) on method: " + method);

}

**@After Advice – Tips**:

* The @After advice does not have access to the exception
  + If you need exception, then use @AfterThrowing advice
* The @After advice should be able to run in the case of success or error
  + Your code should not depend on happy path or an exception
  + Logging / auditing is the easiest case here

**Development Process - @After (Step-by-Step)**:

1. Prep work
2. Add @After advice
3. Test for failure/exception case
4. Test for success case

43. AOP @After Advice Type