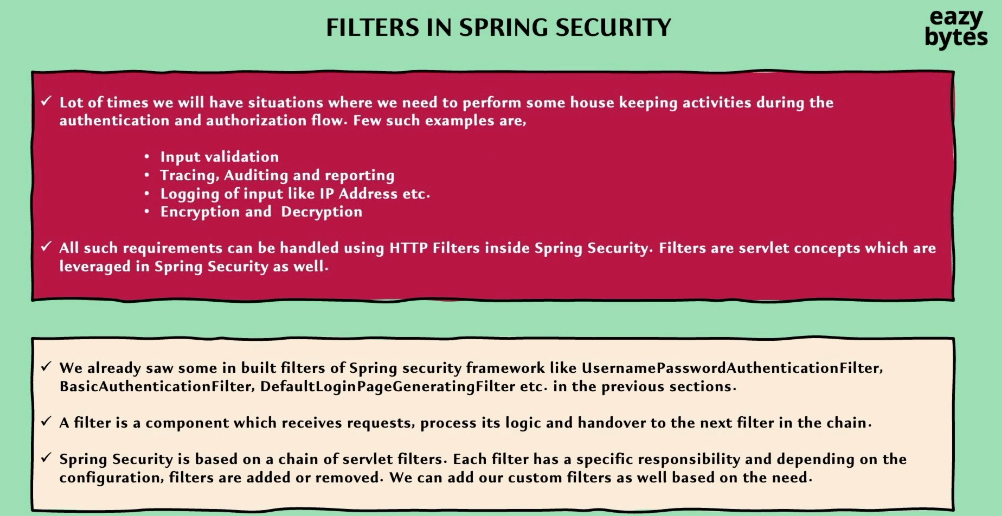
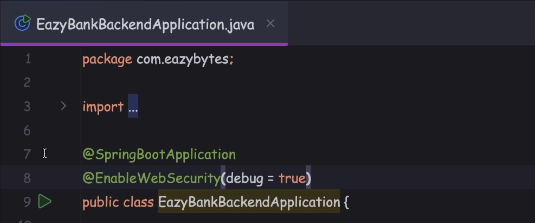
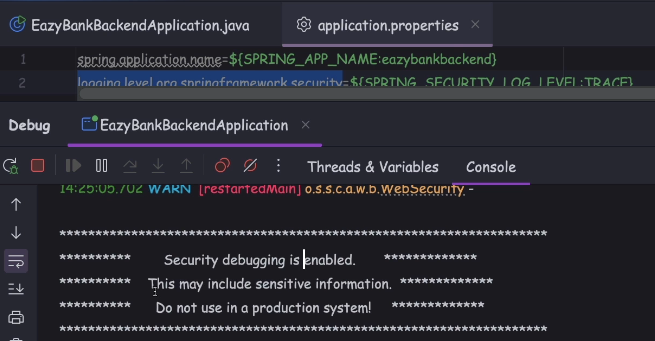
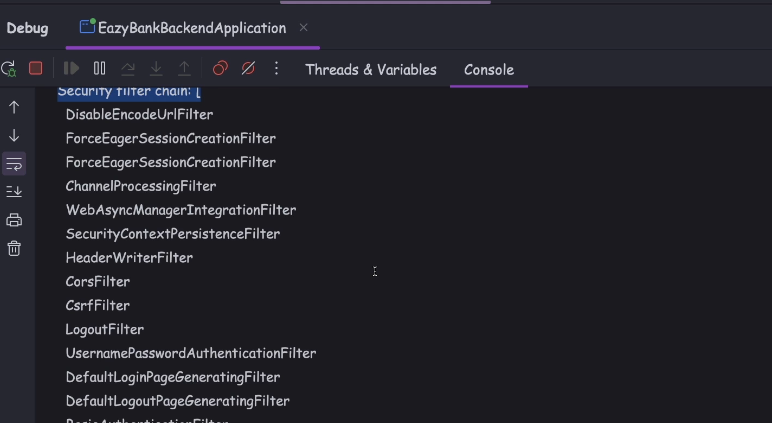
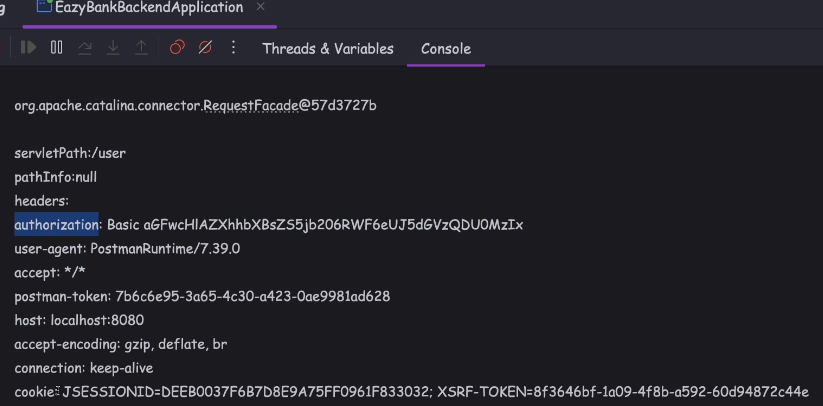
**9.custom filters in spring security**









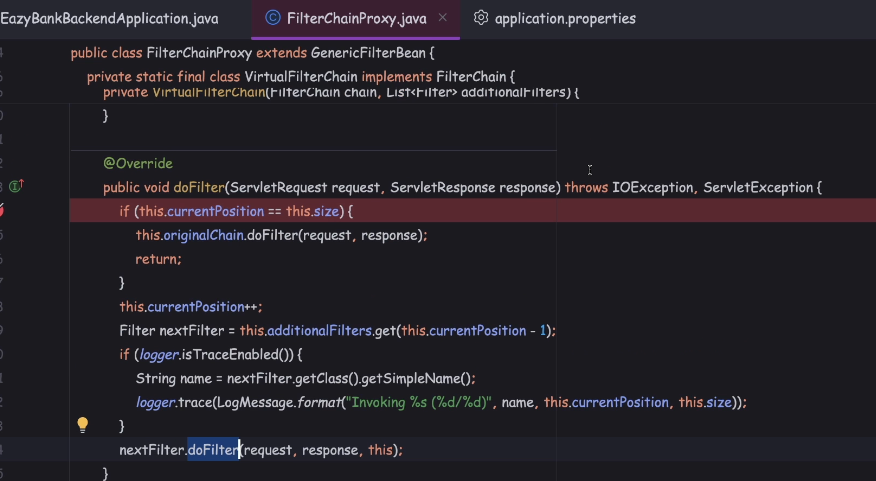


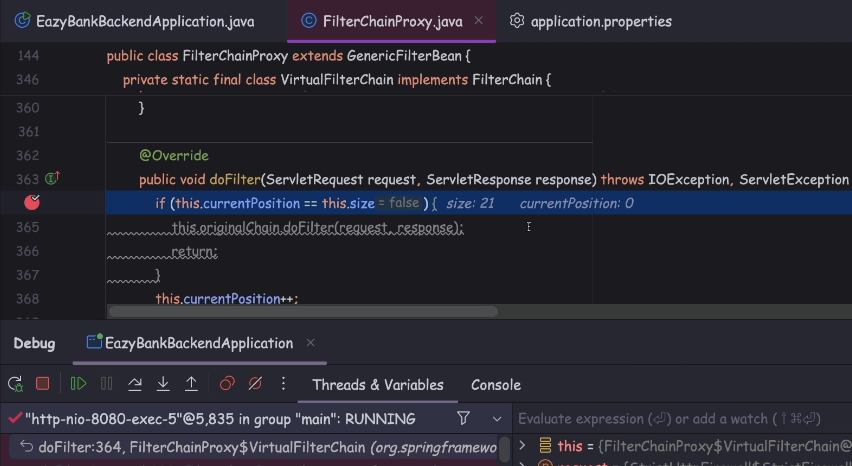
you should be able to see all the Spring Security filters that got executed

as part of your request processing in this format. If you scroll up, you'll be able to see other sensitive information related to the security, like what is the authorization,

headers that I have received, what is the JSESSIONID cookie?

What is a CSRF token cookie?





That means 21 different security filter chains, they are going to be executed as part of my request. So right now, the current portion is zero. So that's why if this if condition is not satisfied, all our Spring Security related filters, they are going to be executed one by one with the logic that we have here.

So if I try to release a break point multiple times,

this time, you'll be able to see current position is

1, 2, 3, 4, 5, 6, 7, 8.

You'll be able to see the filter name also here.

So let me release a break point.

This time, 9, 10, 11, 2, 12, 13, 14, 15,

16, 17, 18, 19, 20, 21.

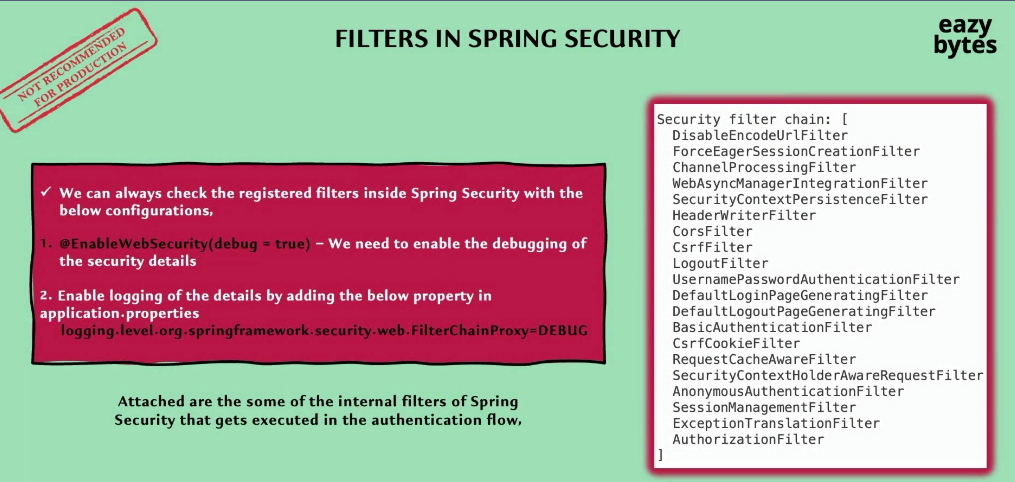
Once all the filters are executed,

this if condition is going to be satisfied. And with that, the request will be forwarded

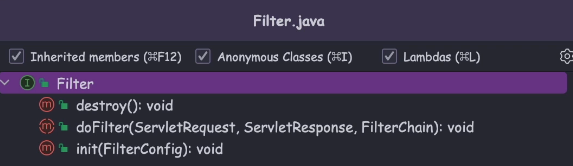
to the original filter of the web application.

From this original filter, the request will be forwarded to the DispatcherServlet

and from this DispatcherServlet, it will reach to the controller layer.



How to create and configure our own custom filter



So this Filter interface is not specific to the Spring Boot library

or this is not specific to the Spring Security library.

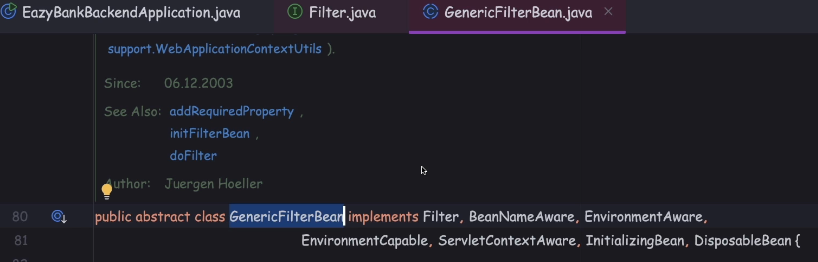
This Filter interface is available from initial days of Java. So this is the option one

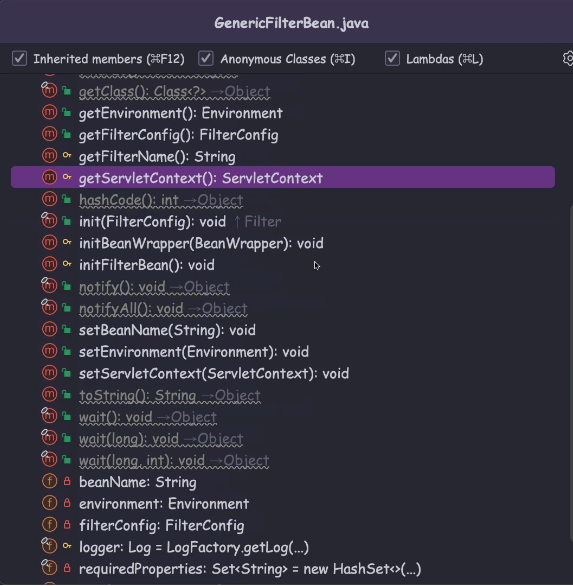
The second option is, GenericFilterBean. So there is an abstract class

with the name GenericFilterBean. This class is not from the Spring Security library,

it is from the Spring Boot or Spring library itself.

By extending this GenericFilterBean, you should be able to define your own custom filter.





Whenever you have a requirement that you want to read some servlet related init parameters that are defined inside your deployment descriptor

like web.xml or if you're looking for an option to read the servlet context details

or the environment property details, so in all such scenarios you can leverage

these abstract class.

If your business logic is a simple business logic, which does not have any dependency on any other parameters that we have discussed as part of GenericFilterBean, then you can happily go with the Filter approach.

Otherwise go with the GenericFilterBean approach.

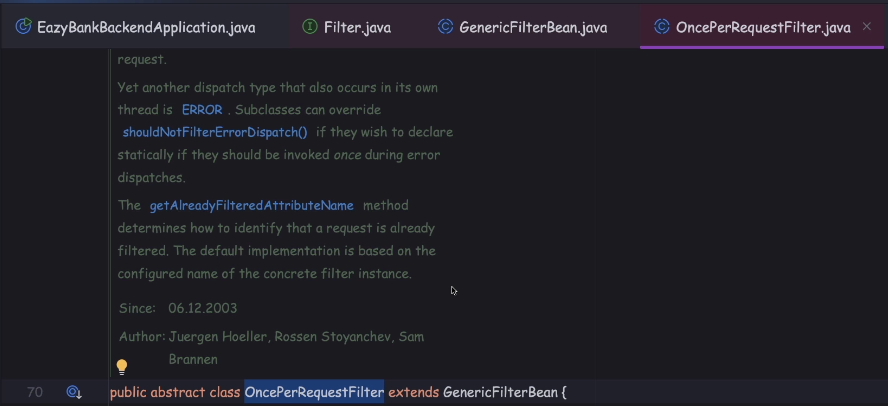
Apart from these two approaches, there is one more approach, which is the recommended approach in certain scenarios.

So this approach we can build with the help of OncePerRequestFilter.

This filter is also available inside the Spring libraries itself.

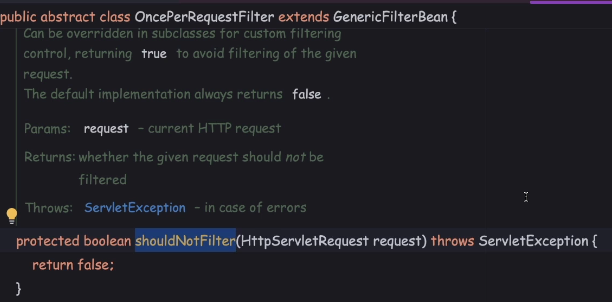
And this abstract class which is OncePerRequestFilter, you should be able to see

that it is extending the GenericFilterBean.



the Spring framework, it is going to guarantee you

that this filter is going to be executed only once at max for each request.



So inside this doFilterInternal() method only, you need to write all your business logic.

This OncePerRequestFilter it also has other useful methods like shouldNotFilter().

This method you can use in the scenarios whenever you want your filter not

to be executed under certain scenarios, maybe you may have a requirement

that your filter should not be executed for certain MVC parts or for certain rest API parts.

So all such logic you can override inside this method and return true from this method, once your conditions are satisfied. So whenever this method returns true,

then your filter is not going to be executed otherwise it is always going to be executed for all type of requests.

**"How to inject our own custom filter into the Spring Security filter chain."**

**Similarly, we also have other methods like addFilter(), addFilterBefore(), addFilterAt().** We need to use one of these methods to configure our own custom filter based upon our business requirements.

.addFilterAfter(new CsrfCookieFilter(), BasicAuthenticationFilter.class)

.addFilterBefore(new CsrfCookieFilter(), BasicAuthenticationFilter.class)

.addFilterAt(new CsrfCookieFilter(), BasicAuthenticationFilter.class)

.addFilter(new CsrfCookieFilter()) // based on order in the filter

addFilterAfter: CsrfCookieFilter filter is going to be executed once the BasicAuthenticationFilter execution is completed.

addFilterBefore: my custom filter is going to be executed just before the BasicAuthenticationFilter

addFilterAt: This configuration will execute your custom filter at the same portion

where the BasicAuthenticationFilter is executed.

In this kind of configuration, the Spring Security framework, it is going

to randomly choose one of the filter. So sometimes it is going to execute this custom filter as first followed by BasicAuthenticationFilter and vice versa.

So whenever we are using addFilterAt() both of the filters that you passed as an input, they're going to be executed at the same portion.

But the execution of these two filters internally can't be controlled by us.

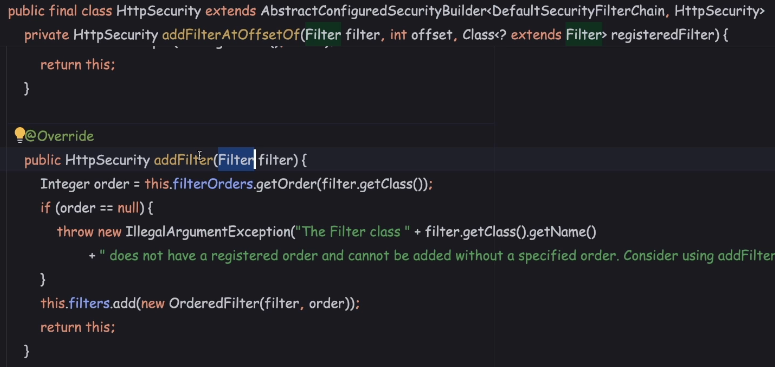
addFilter we should only pass our own custom filter but not the second parameter.

we need to make sure we are mentioning the order to our filter by using order annotation.

On top of our custom filter we can mention @Order annotation

and to this @Order annotation, we can give a number like 9, 10, based upon the order number that I have mentioned inside my custom filter.

My custom filter is going to be added into the list at a given position.



If you don't mention this @Order annotation, then obviously you're going to get a runtime exception.

But I don't recommend you using this order because sometimes what is going

to happen is we might have calculated the order by looking at the order of execution of the internal provided filters.

So these internal provided filters, orders may keep changing from version to version

and that may result into unexpected issues if you try to hard code a specific order inside our custom filter.

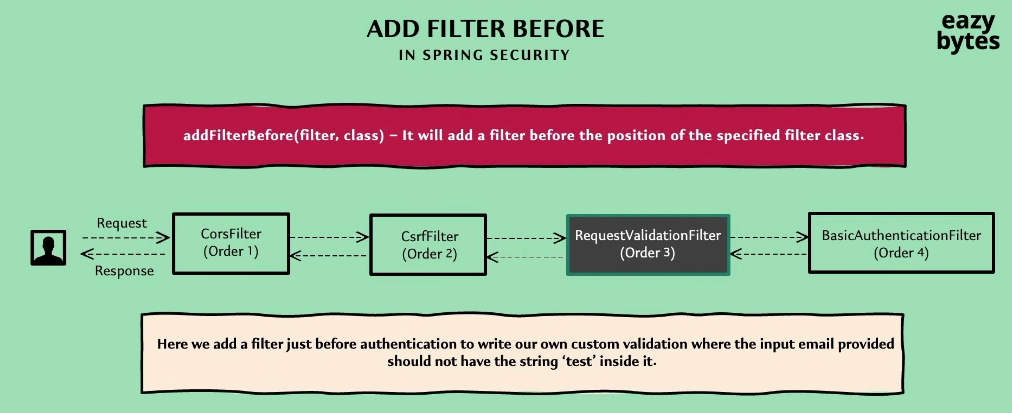
So that's why I don't use this method.

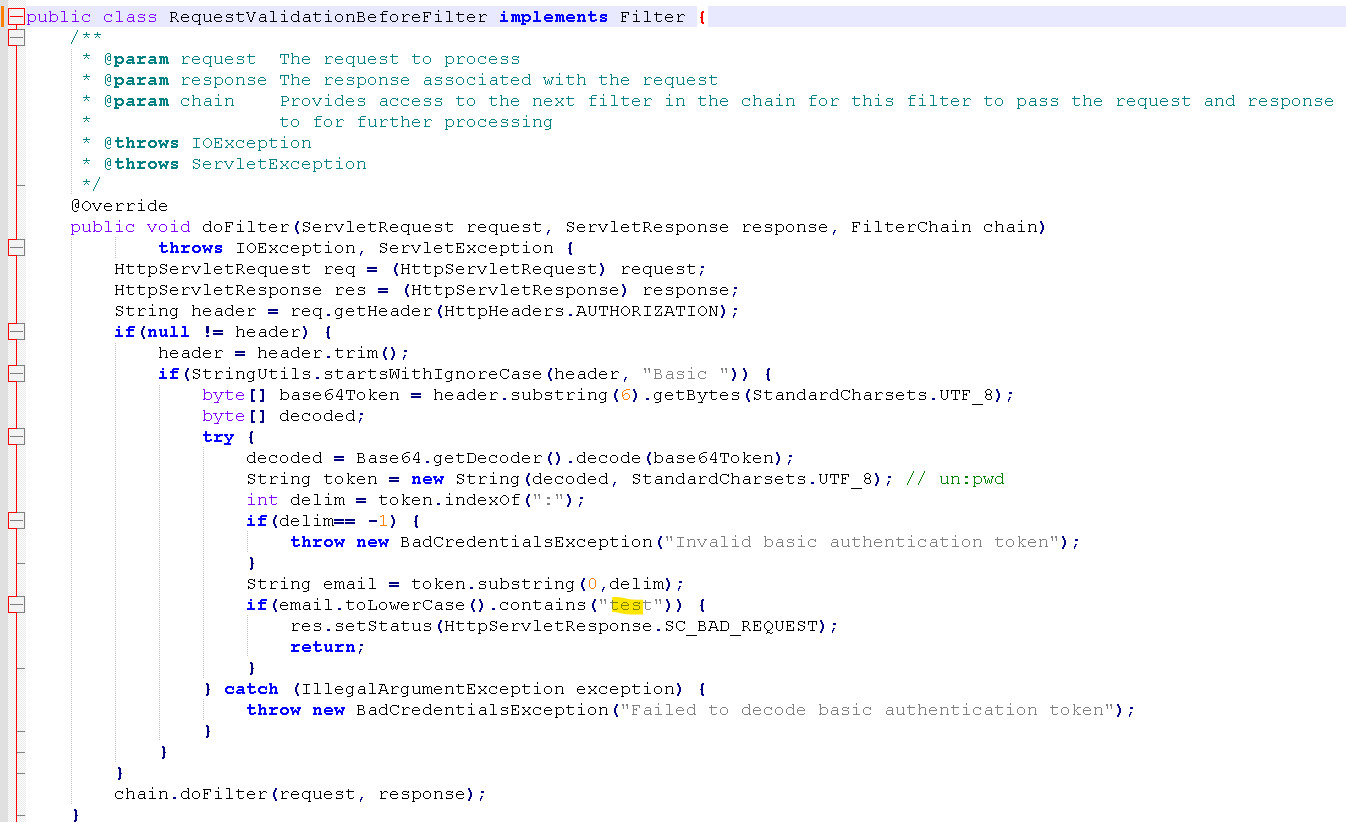
We are only going to discuss three methods,

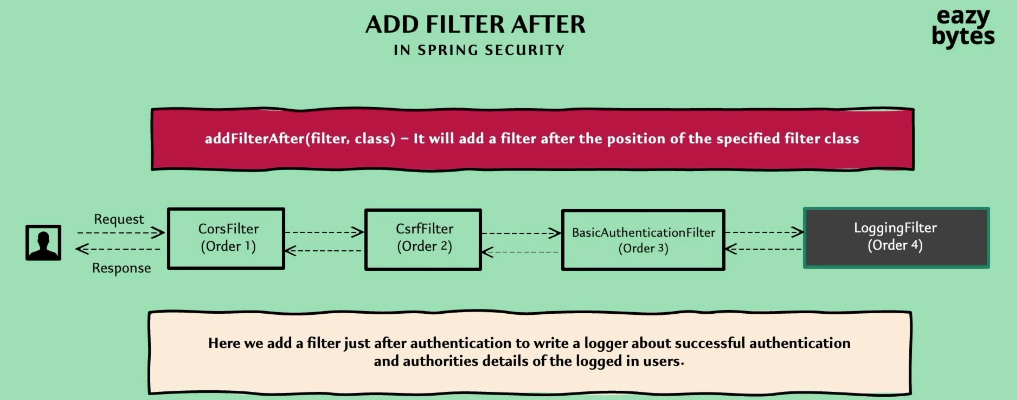
which are addFilterAt(), addFilterBefore() and addFilter().

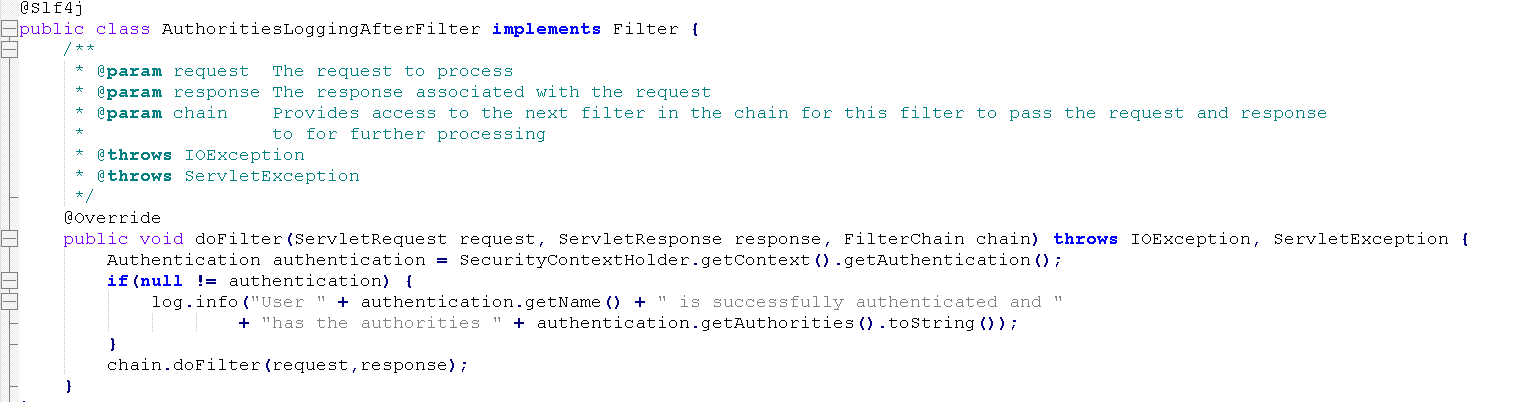


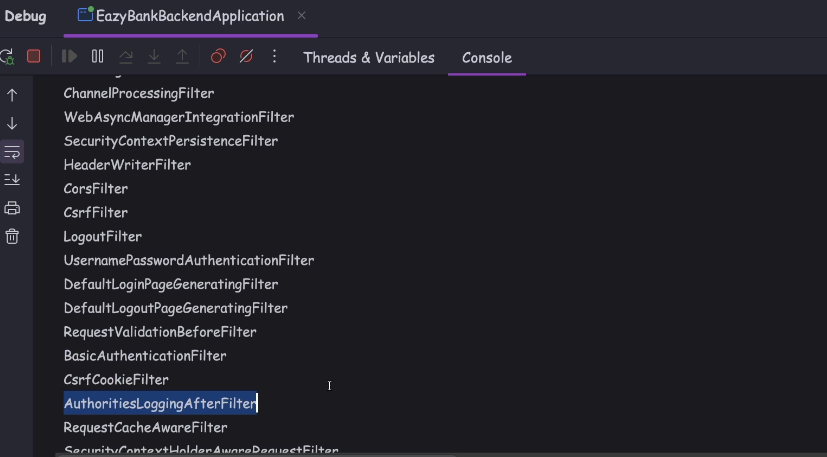
Adding a custom filter using addFilterBefore()

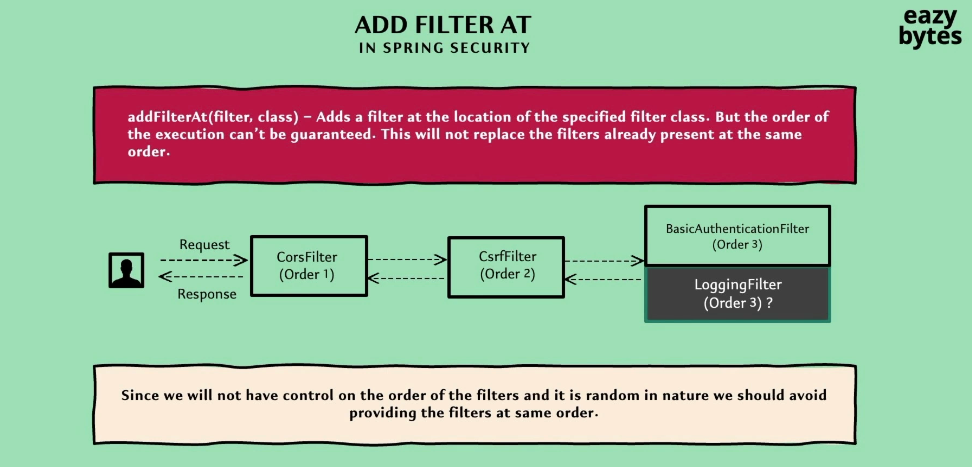


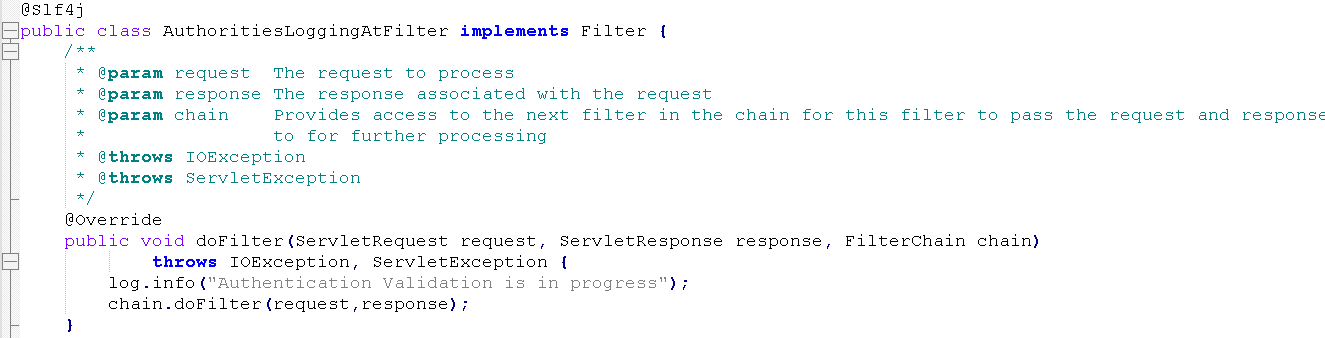












Top of Form

In order to create our own custom filters in Spring Security, we need to implement the following interface ?

**jakarta.servlet.Filter**

Top of Form

If we have a scenario to define a custom filter with the name 'RequestValidationBeforeFilter' and run it before a specified filter 'BasicAuthenticationFilter' as part of Spring Security authentication and authorization, then which of the below code is correct configuration ?

**addFilterBefore(new RequestValidationBeforeFilter(), BasicAuthenticationFilter.class)**

Top of Form

If we have a scenario to define a custom filter with the name 'JWTTokenGeneratorFilter' and run it after a specified filter 'BasicAuthenticationFilter' as part of Spring Security authentication and authorization, then which of the below code is correct configuration ?

**addFilterAfter(new JWTTokenGeneratorFilter(), BasicAuthenticationFilter.class)**

Top of Form

If we have a scenario where we need to make sure to execute our custom filter only a single execution per request dispatch then which of the following filter needs to be extended inside our custom filter ?

**org.springframework.web.filter.OncePerRequestFilter**