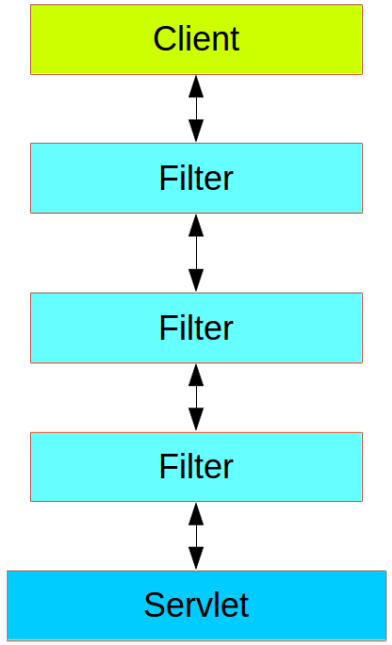
Servlet Filters and Chains Outline

- Servlet Filters
- Order of the Filter Chain - I
- Order of the Filter Chain – II
- FilterChainProxy – I
- FilterChainProxy – II
- FilterChainProxy – III
- DelegatingFilterProxy
- SecurityFilterChain

Servlet Filters

Spring Security in the web tier is based on Servlet Filters.



Typical layering of the handlers for a **single** HTTP request.

The client sends a request to the application → The container decides which filters and which servlet apply to it based on the path of the request URI.

At most, one servlet can handle a single request, but ***filters form a chain***, so they are **ordered**. → In fact, a filter ***can veto the rest of the chain*** if it wants to handle the request itself.

A filter ***can also modify*** the request, or the response used in the downstream filters and servlet.

Order of the Filter Chain - I

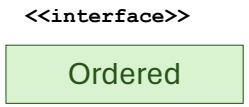
- The **order** of the filter chain is **very important**, and Spring Boot manages it through two mechanisms:

@Beans of type Filter can

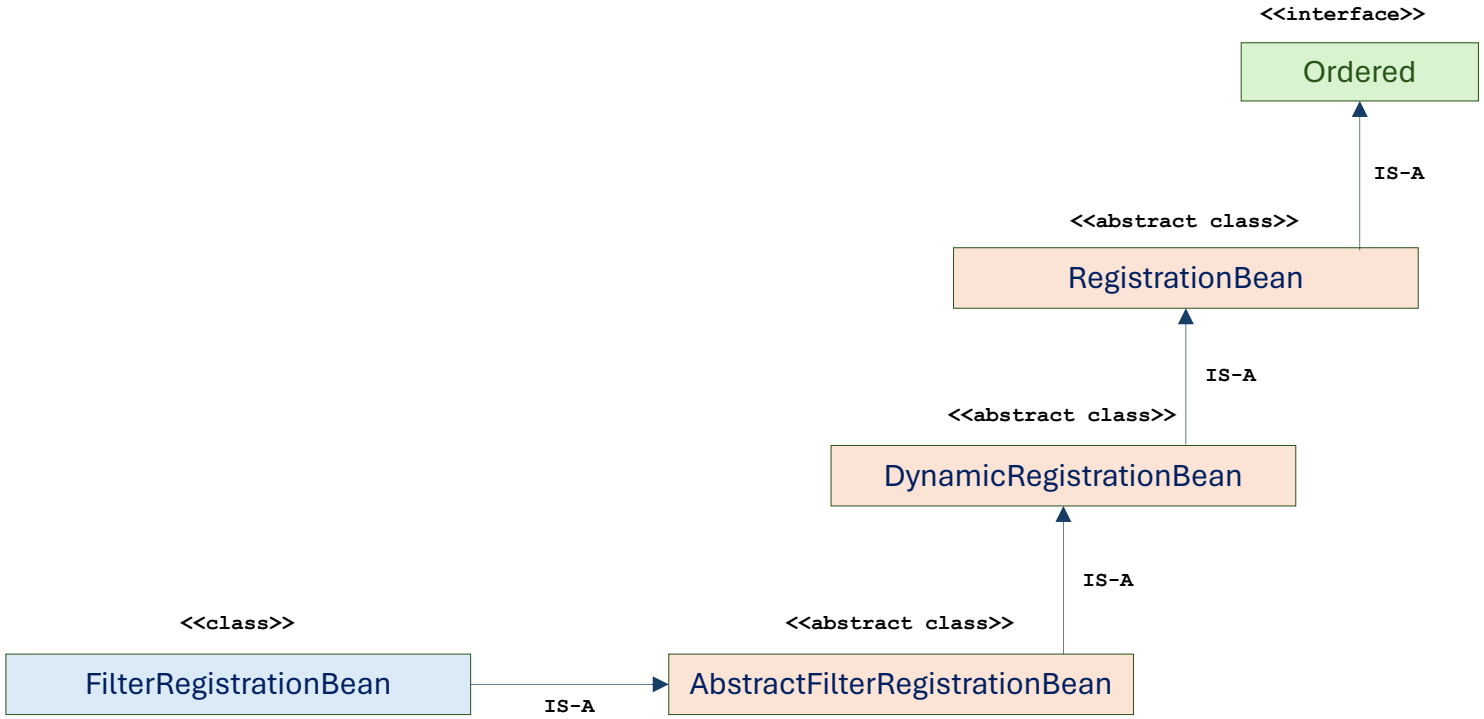
Be annotated by the @Order annotation

Implement the Ordered interface

Be part of a FilterRegistrationBean ***that itself has an order*** as part of its API




```java
public interface Ordered {
    int HIGHEST_PRECEDENCE = Integer.MIN_VALUE;
    int LOWEST_PRECEDENCE = Integer.MAX_VALUE;

    int getOrder();
}
```



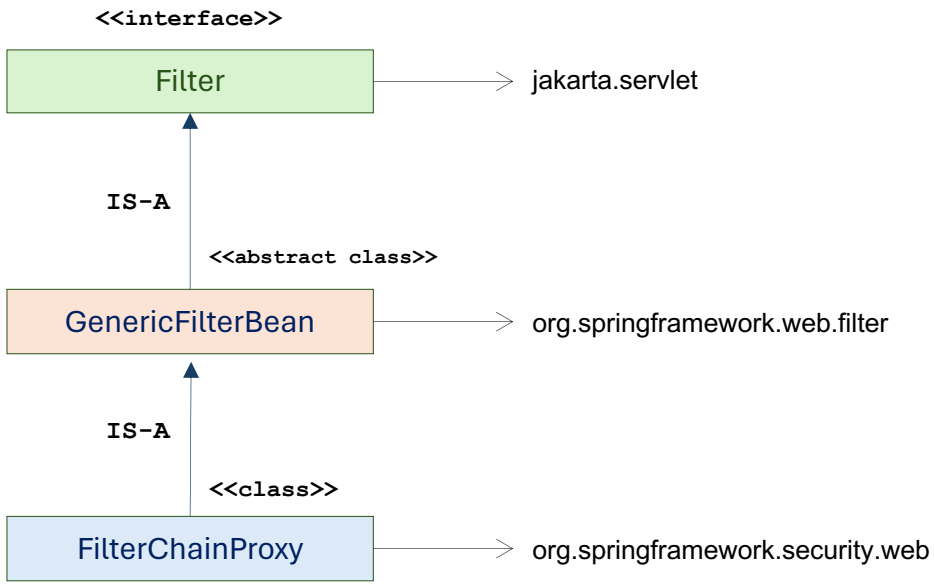
Order of the Filter Chain - II

- Some **off-the-shelf** filters define their own constants to help signal what order they like to be in relative to each other:
  - SessionRepositoryFilter has a DEFAULT_ORDER of Integer.MIN_VALUE + 50,
  - which tells us it likes to be early in the chain, but it does not rule out other filters coming before it.


```
@Order(SessionRepositoryFilter.DEFAULT_ORDER)
public class SessionRepositoryFilter<S extends Session> extends OncePerRequestFilter {
    . . . // intentionally skipped

    public static final int DEFAULT_ORDER = Integer.MIN_VALUE + 50;

 . . . // intentionally skipped
}
```

FilterChainProxy - I

- Spring Security is installed as a single Filter in the chain, and its concrete type is FilterChainProxy.




```
public interface Filter {
    default void init(FilterConfig filterConfig) throws ServletException {
    }

    void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException;

    default void destroy() {
    }
}
```

```
package jakarta.servlet;

import java.io.IOException;

public interface FilterChain {
    void doFilter(ServletRequest request, ServletResponse response)
    throws IOException, ServletException;
}
```

FilterChainProxy - II

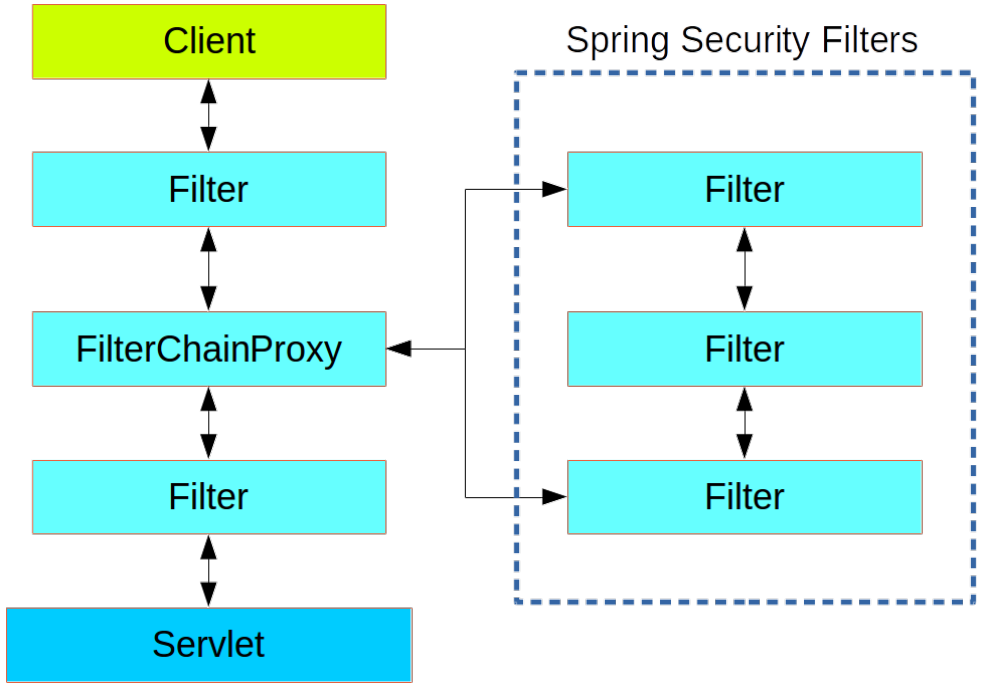
- In a Spring Boot application, the security filter is a @Bean in the ApplicationContext, and it is ***installed by default*** so that it is applied to every request.
    - It is **installed at a position** defined by SecurityProperties.**DEFAULT_FILTER_ORDER** which in turn is anchored by OrderedFilter.**REQUEST_WRAPPER_FILTER_MAX_ORDER**.
    - OrderedFilter.**REQUEST_WRAPPER_FILTER_MAX_ORDER** is the maximum order that a Spring Boot application expects filters to have if they wrap the request, modifying its behavior.

```
package org.springframework.boot.web.servlet.filter;

public interface OrderedFilter extends Filter, Ordered {
    int REQUEST_WRAPPER_FILTER_MAX_ORDER = 0;
}
```

```
package org.springframework.boot.autoconfigure.security;

public class SecurityProperties {
    . . . // intentionally skipped

    public static final int DEFAULT_FILTER_ORDER = OrderedFilter.REQUEST_WRAPPER_FILTER_MAX_ORDER -100;

    . . . // intentionally skipped
}
```

FilterChainProxy - III

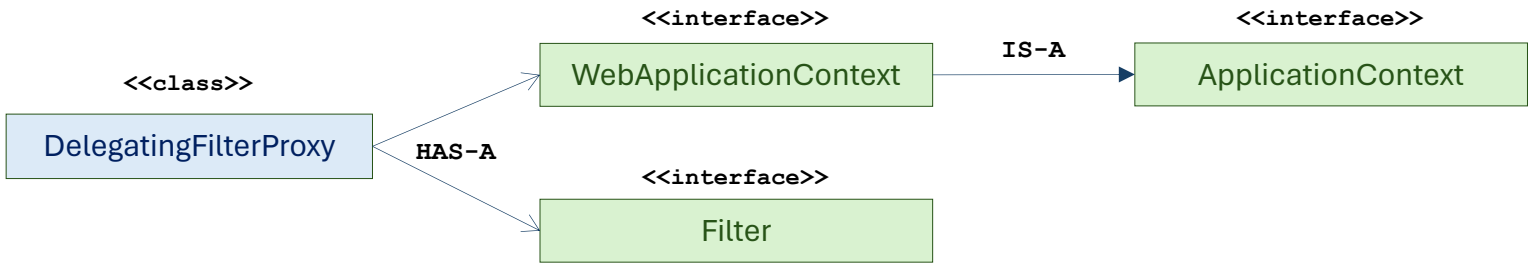
- From the point of view of the container, **Spring Security** is a single physical Filter, but, **inside of it**, there are additional filters, each playing a special role.
    - delegates **processing** to a chain of internal filters.

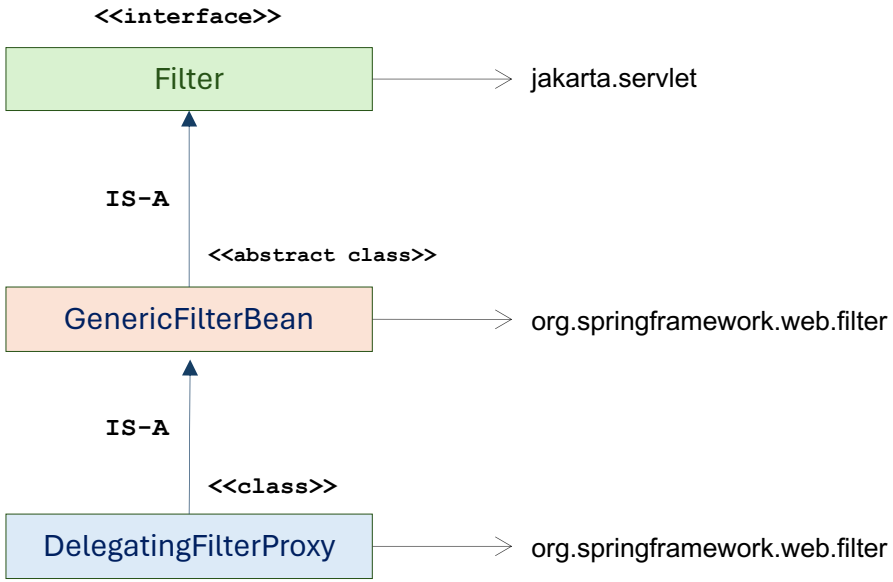


- It is the FilterChainProxy that **contains all the security logic** arranged internally as a **chain** (or chains) **of filters**.
    - All the filters have the **same API** since they all implement the Filter interface.
    - They all have the opportunity to **veto** the rest of the chain.

DelegatingFilterProxy

- There is even one more **layer of indirection** in the security filter
  - usually installed in the container as a DelegatingFilterProxy, which **does not have** to be a Spring @Bean.
  - delegates to a FilterChainProxy, which is **always** a @Bean, usually with a fixed name of **springSecurityFilterChain**.

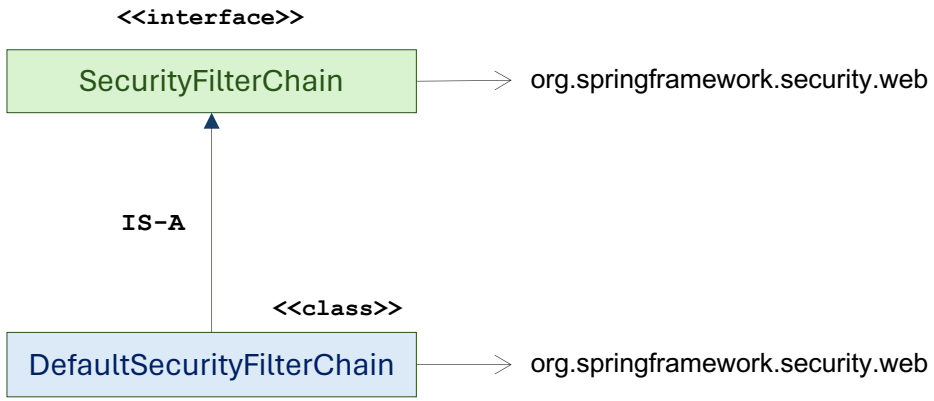


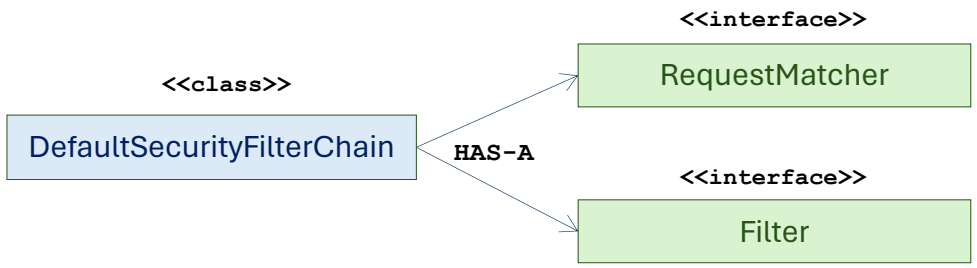



```
package org.springframework.boot.autoconfigure.security;

public class DelegatingFilterProxy extends GenericFilterBean {
    . . . // intentionally skipped

    private WebApplicationContext webApplicationContext;

    private volatile Filter delegate;

    . . . // intentionally skipped
}
```

SecurityFilterChain

- Defines a **filter chain** which is capable of being matched against an HttpServletRequest to decide whether it applies to that request.
- Used to configure a FilterChainProxy.






```java
package org.springframework.security.web;

public interface SecurityFilterChain {
    boolean matches(HttpServletRequest request);

    List<Filter> getFilters();
}
```

```java
package org.springframework.security.web;

public interface DefaultSecurityFilterChain implements SecurityFilterChain {
    . . . // intentionally skipped

    private final RequestMatcher requestMatcher;

    private final List<Filter> filters;

    . . . // intentionally skipped
}
```