Spring Security Framework

Harezmi Bilişim Çözümleri

Ajanda

- Güvenlik İhtiyaçlarına Genel Bakış
- Spring Security Nedir? & Özellikleri
- Neden Spring Security?
- Spring Security Yapıtaşları
- Spring Security Filter Zinciri
- Kimliklendirme & Kimliklendirme Modeli
- Başarılı ve Başarısız Login Akış Örnekleri

Ajanda

- Yetkilendirme
- Yetkilendirme Akışı & Yetkilendirme Modeli
- Namespace Konfigürasyonu
- Namespace ile Spring Security Kullanımı

Güvenlik İhtiyaçlarına Genel Bakış

- Kurumsal uygulamaların 2 temel güvenlik ihtiyacı söz konusudur

1. Kimliklendirme (Authentication)
2. Yetkilendirme (Authorization)
   - Web isteklerinin yetkilendirilmesi
   - Servis metodlarının yetkilendirilmesi
   - Domain nesnelerine yetki bazında erişim sağlanması
   - Nesnelerin attribute'ları düzeyinde erişim

Spring Security Nedir?

- Kurumsal Java uygulamaları için authentication ve authorization servisleri sağlayan bir framework
- 2004 yılında Spring üzerine kurulu Acegi Security Framework adı ile ortaya çıkmıştır
- 2007 yılına gelindiğinde Spring ürün portföyünde yer almıştır

Özellikleri

- Pek çok değişik kimliklendirme (authentication) yöntemini desteklemektedir
- Form tabanlı, basic, digest, NTLM, Kerberos, JAAS...
- CAS, OpenID, Siteminder gibi SSO çözümleri ile entegre çalışabilmektedir
- Kullanıcı bilgisinin değişik ‘realm’lerdan (Memory, LDAP, JDBC...) alınmasını sağlar

Özellikleri

- Çalışma zamanında dinamik olarak kullanıcı bilgisinin değiştirilmesini sağlar
- Aynı kullanıcının sisteme erişim sayısını yönetir
- Otomatik olarak “remember me” desteği sunar
- URL, metod ve domain nesneleri düzeyinde yetkilendirme sağlar

Özellikleri

- RMI ve HttpInvoker gibi remote method çağrılarında kimlik bilgisinin istemciden sunucuya taşınmasını sağlar
- Web Container'ları ile entegre çalışabilir
- Güvenlikli bir HTTP iletişimi kurmaya yardımcı olur
- Güvenlikle ilgili HTTP response header'larını yönetebilir

Neden Spring Security?

- Neden web.xml security değil?
- Neden JAAS değil?

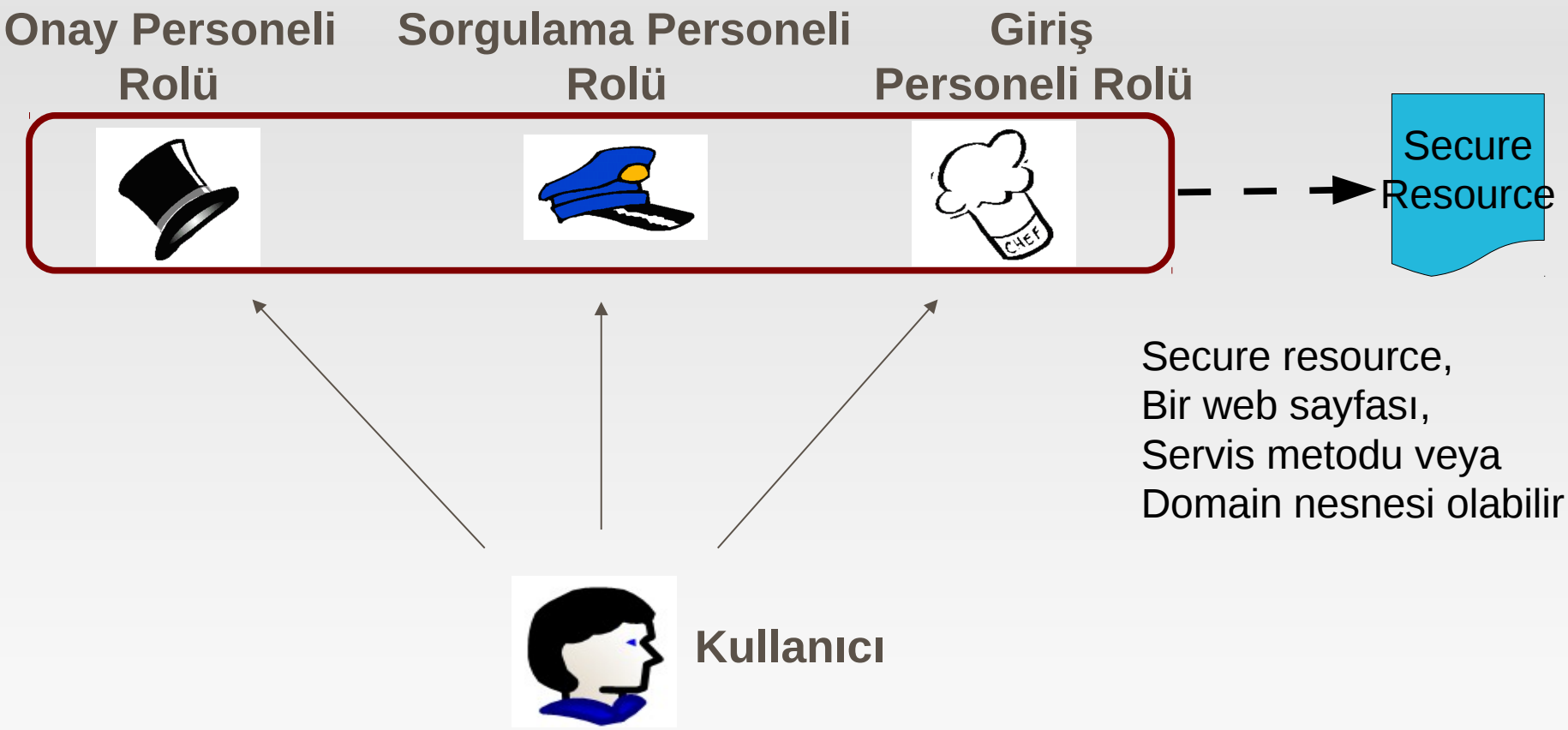
Neden web.xml security değil?

- Web container'a bağımlı bir kimliklendirme altyapısı ortaya çıkar
- Web kaynaklarını(URL) yetkilendirmek için sınırlı bir model mevcuttur
- Uygulamada metod ve domain nesneleri düzeyinde yetkilendirme yapmak mümkün değildir

Neden JAAS değil?

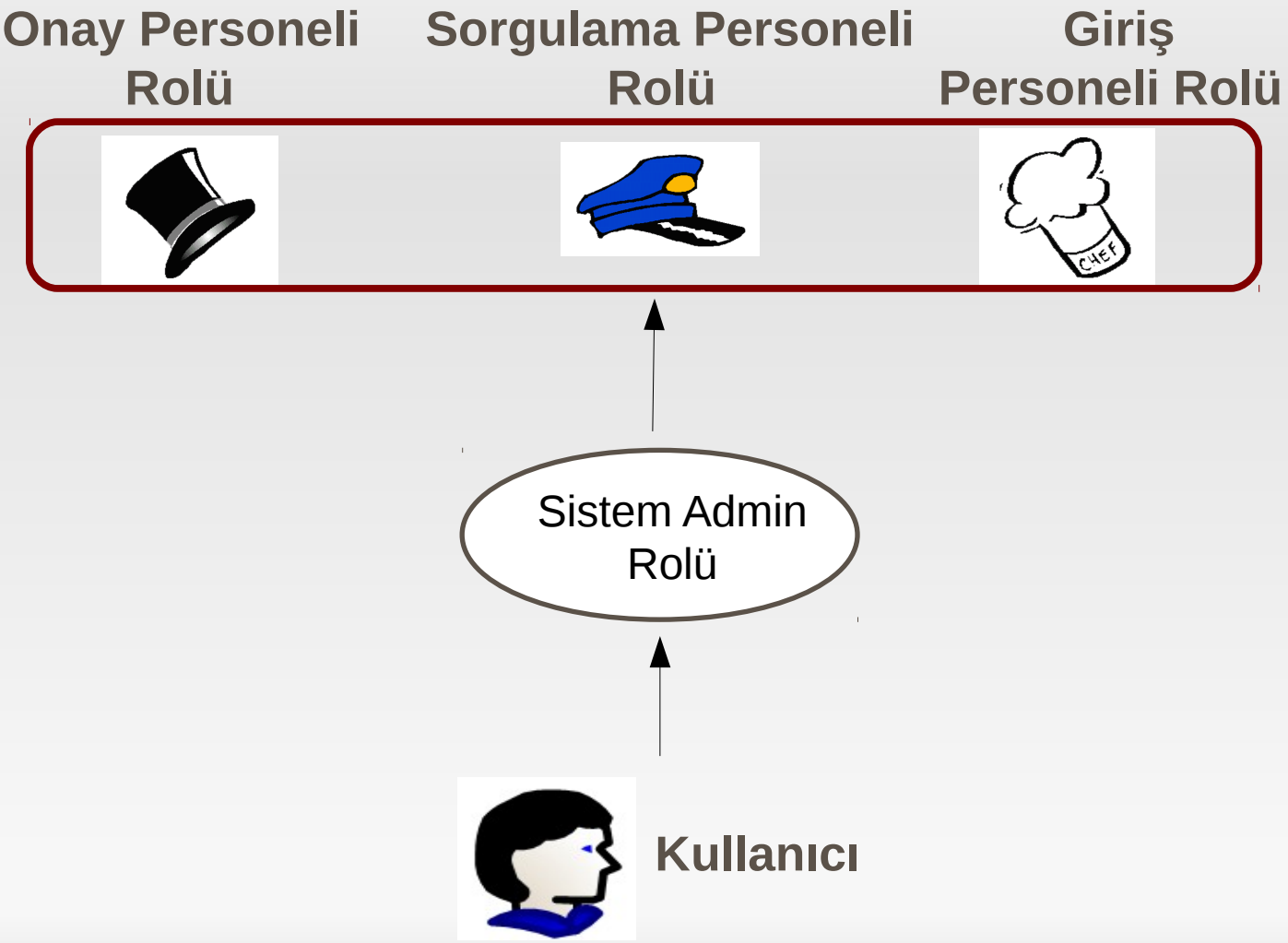
- JRE düzeyinde karmaşık tanımlar yapılmaktadır
- Yine container düzeyinde bağımlılık ortaya çıkmaktadır
- Kimliklendirme tarafı istenirse JAAS'a havale edilebilir

Temel Yapıtaşları: Kullanıcı – Rol İlişkisi

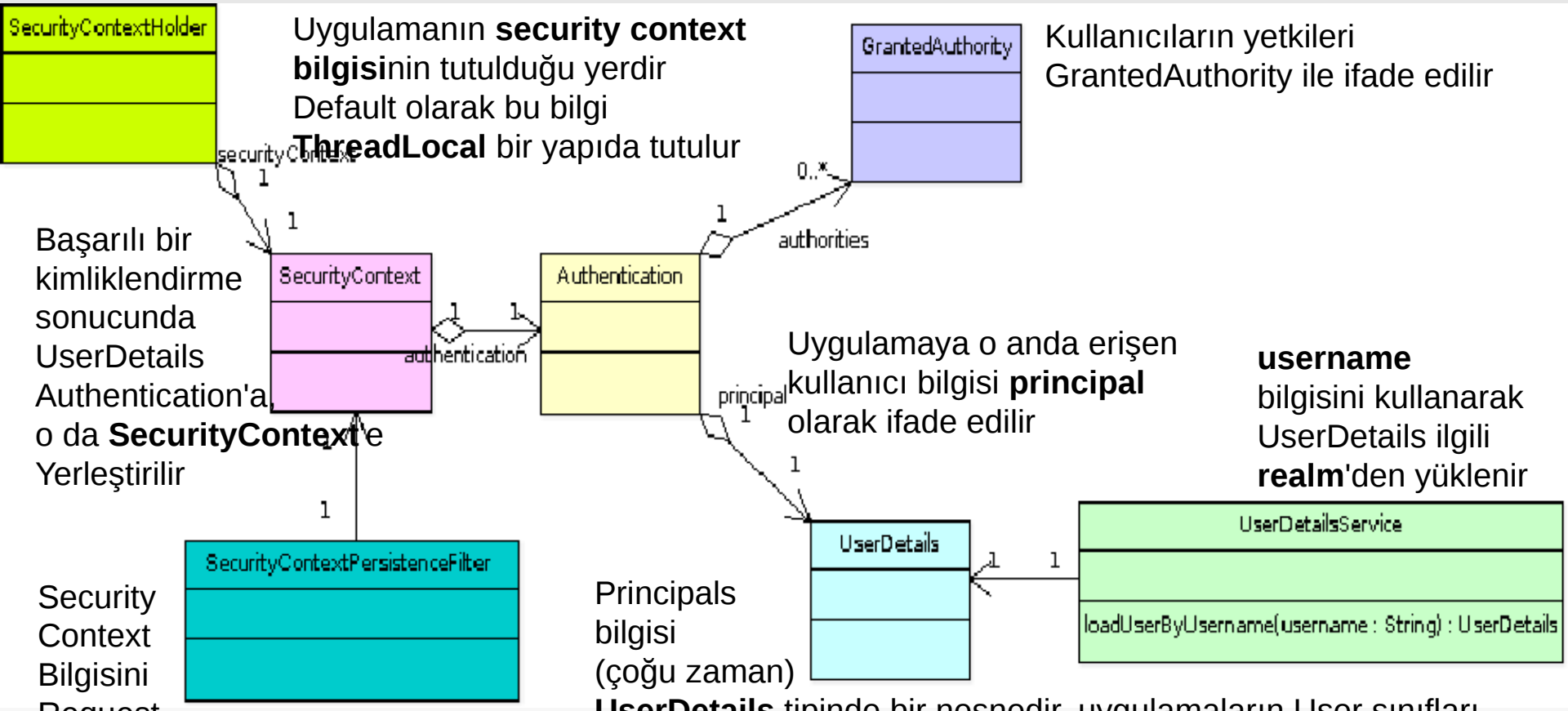


Secure resource,
Bir web sayfası,
Servis metodu veya
Domain nesnesi olabilir

Temel Yapıtaşları : Rol Grubu – Rol İlişkisi



Temel Yapıtaşları: Domain Sınıfları



Uygulamanın **security context bilgisi**nin tutulduğu yerdir
Default olarak bu bilgi **ThreadLocal** bir yapıda tutulur

Kullanıcıların yetkileri GrantedAuthority ile ifade edilir

Başarılı bir kimliklendirme sonucunda UserDetails Authentication'a, o da **SecurityContext**'e Yerleştirilir

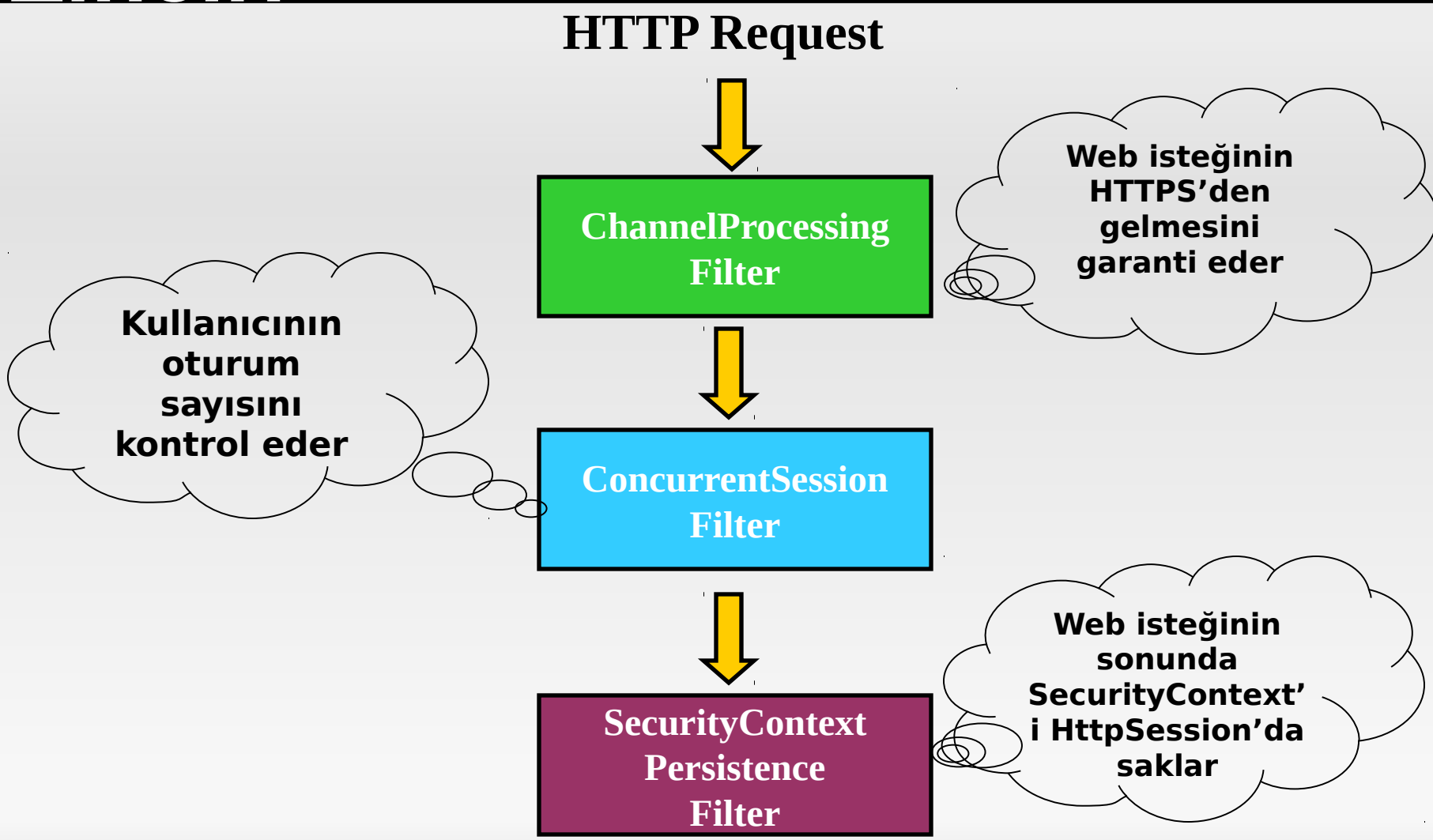
Uygulamaya o anda erişen kullanıcı bilgisi **principal** olarak ifade edilir

username bilgisini kullanarak UserDetails ilgili **realm**'den yüklenir

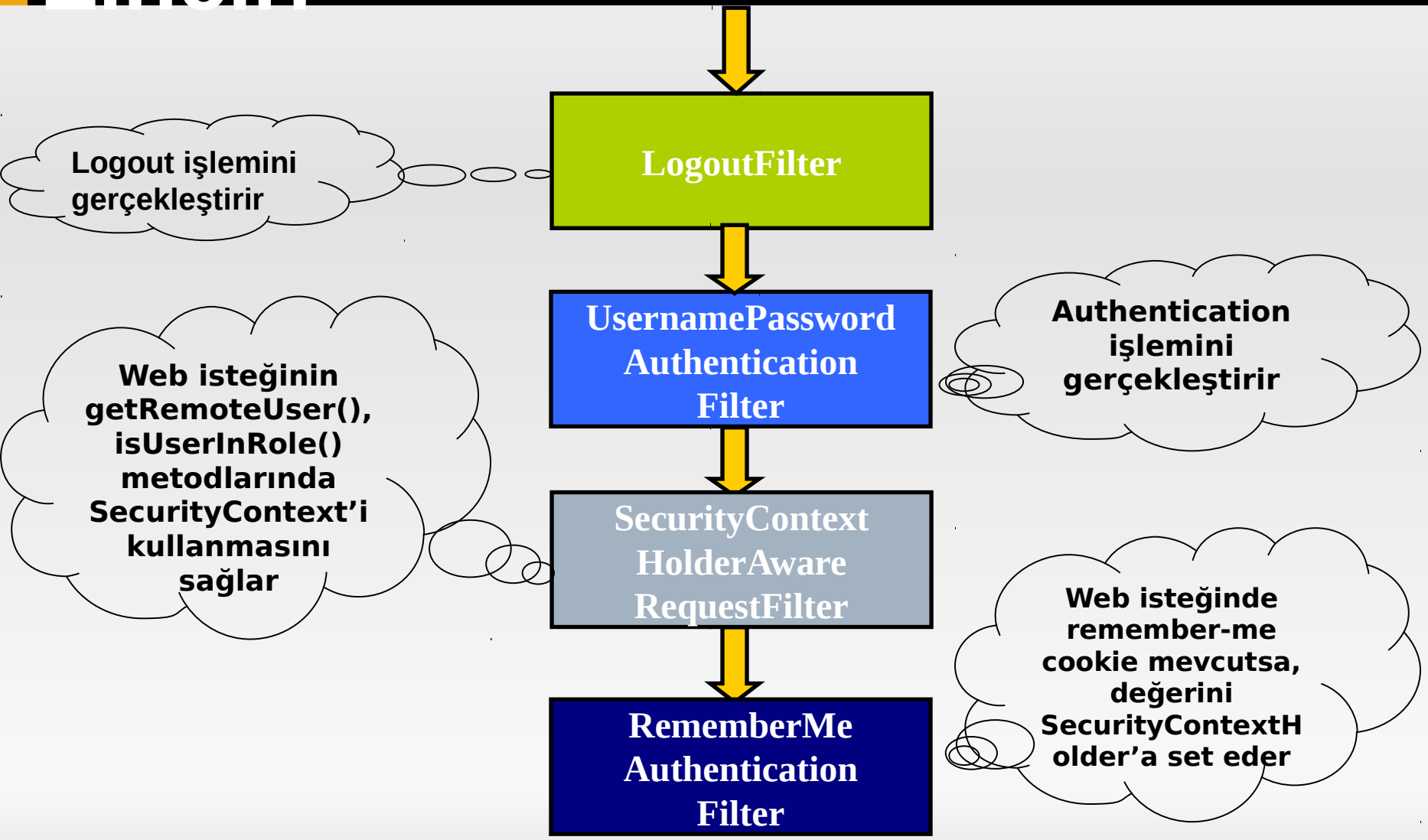
Security Context Bilgisini Request aralıklarında HttpSession'da saklar

Principals bilgisi (çoğu zaman) **UserDetails** tipinde bir nesnedir, uygulamaların User sınıfları bu arayüzü implement eder

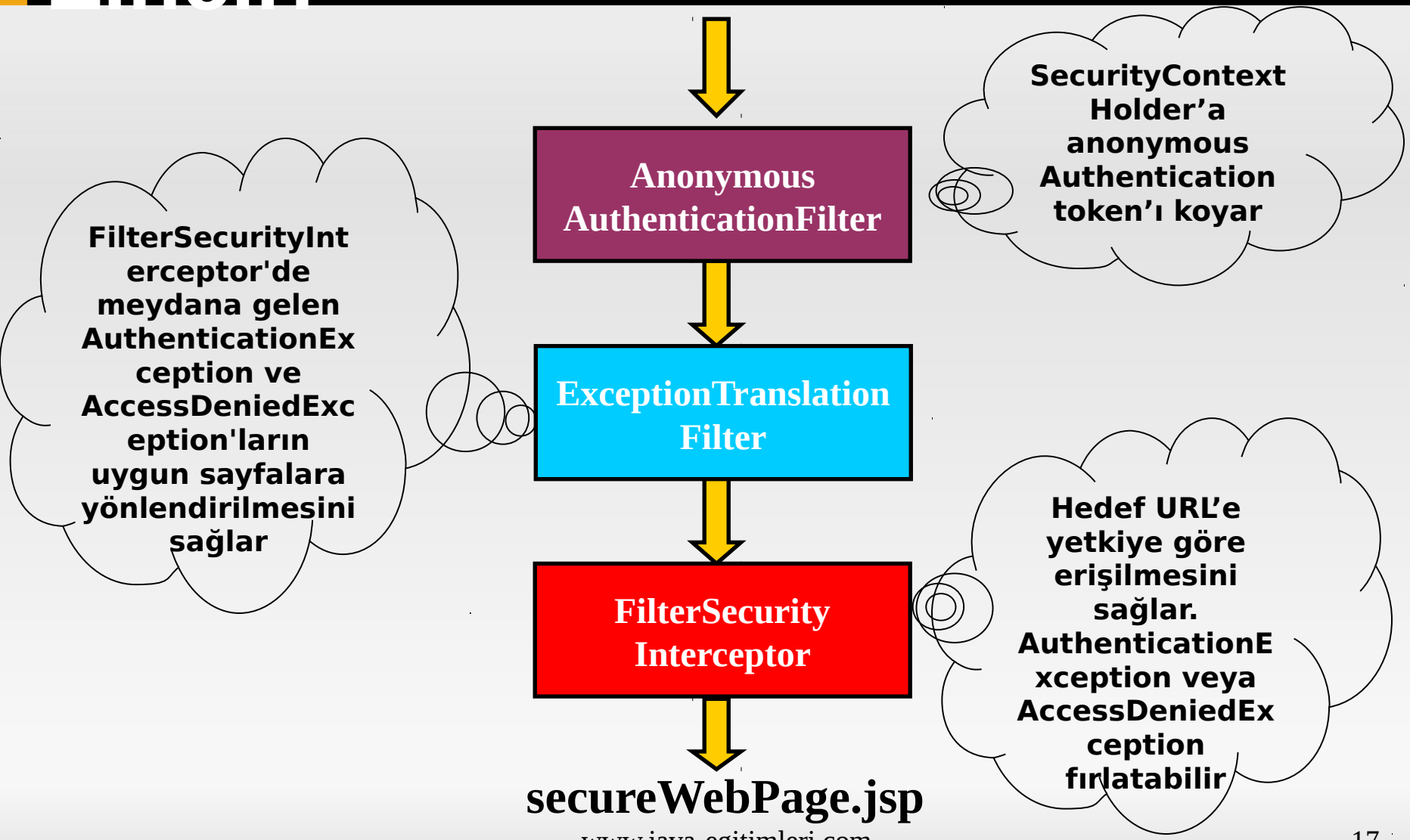
Spring Security Filter Zinciri



Spring Security Filter Zinciri



Spring Security Filter Zinciri



Spring Security Konfigürasyonu

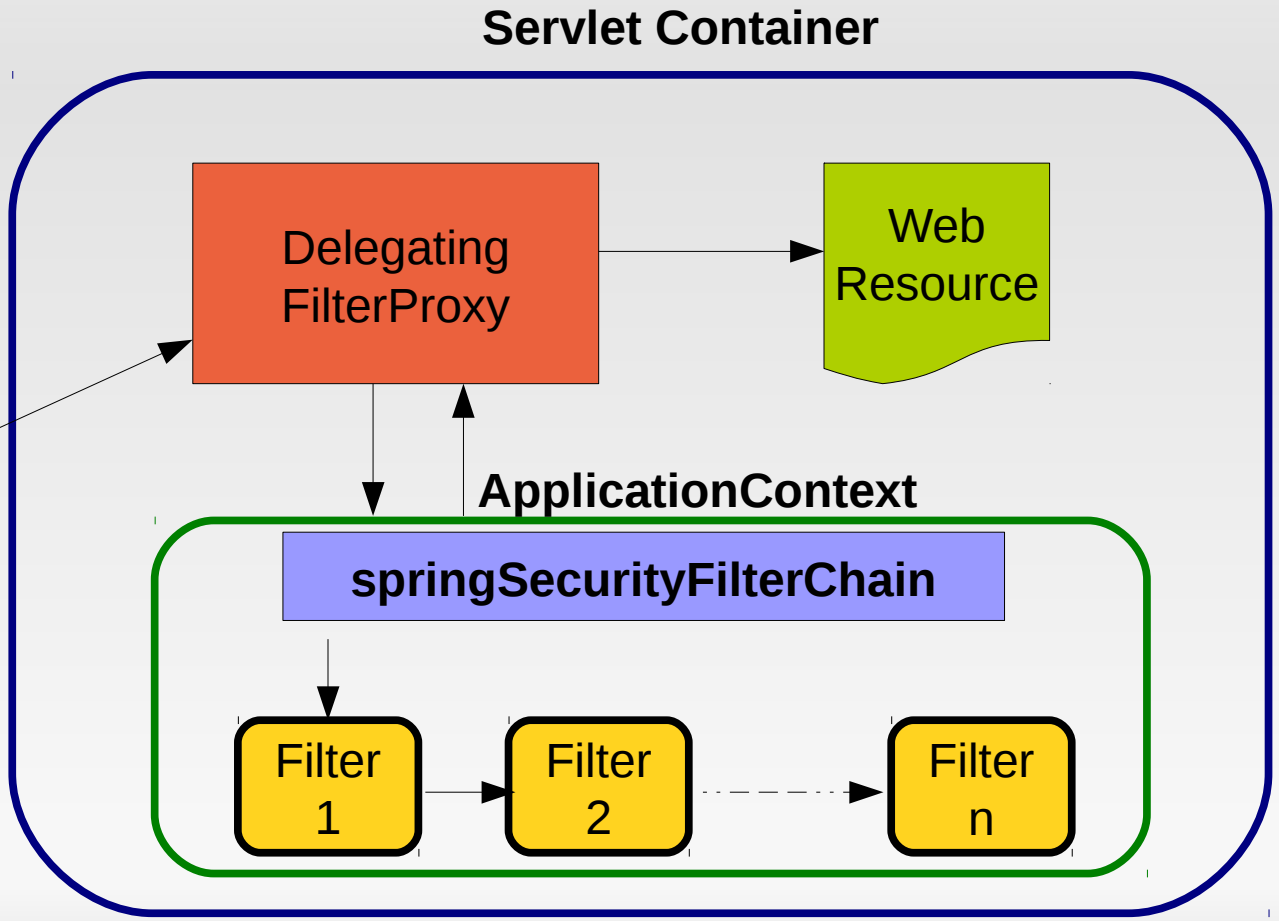
```
<filter>
   <filter-name>
      springSecurityFilterChain
   </filter-name>
   <filter-class>
org.springframework.web.filter.DelegatingFilterProxy
   </filter-class>
</filter>

<filter-mapping>
   <filter-name>
      springSecurityFilterChain
   </filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

web.xml içerisinde **DelegatingFilterProxy** ile Bir Servlet Filter tanımı yapılır

Spring Security Konfigürasyonu

DelegatingFilterProxy sınıfı, web.xml içerisindeki tanım ile ApplicationContext içinde tanımlı Filter bean'ları arasında **köprü vazifesi** görür

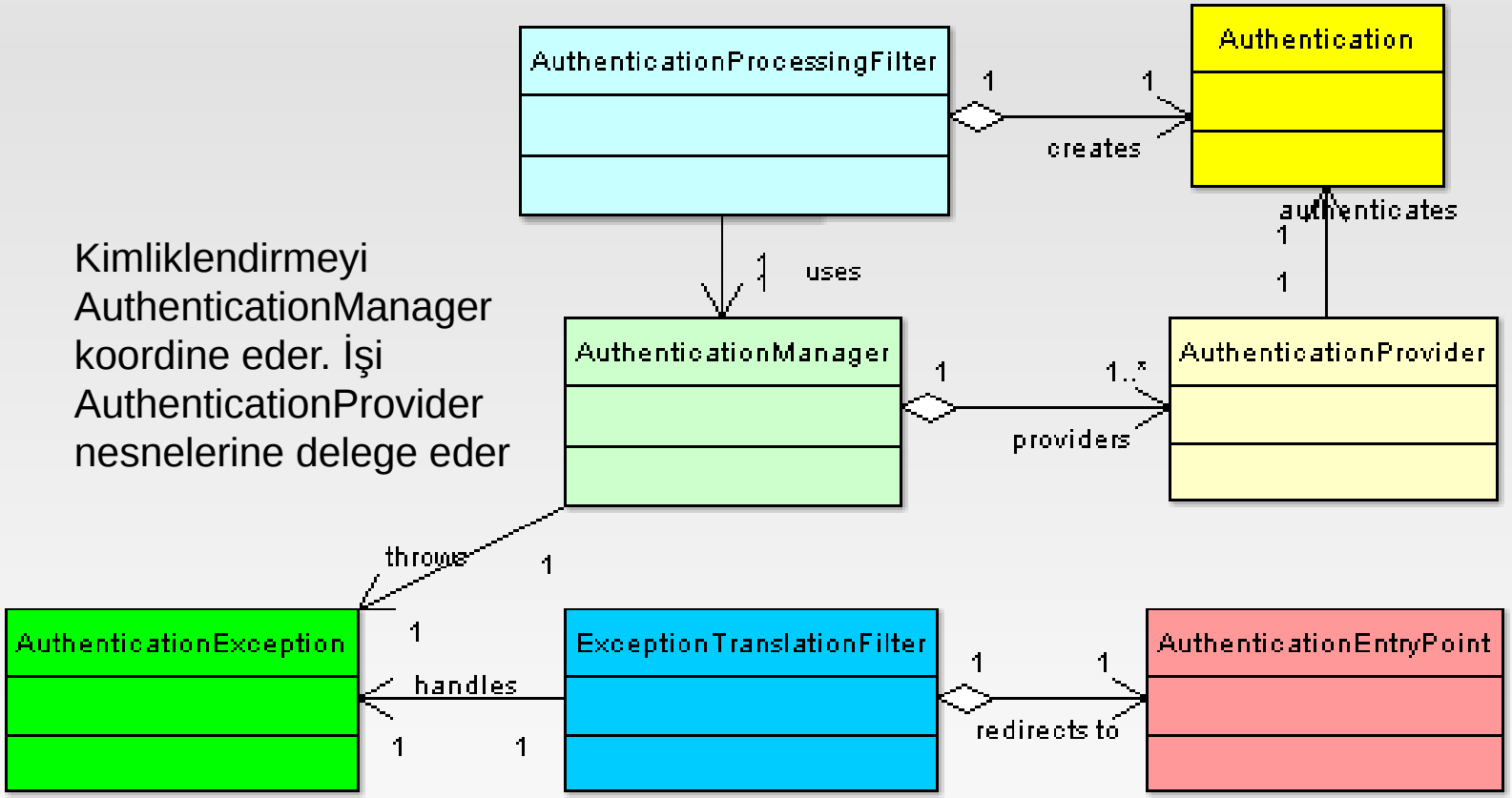
FilterChainProxy'de web.xml içerisinde **tek bir filter** ile bütün security filter zincirinin tanımlanmasını sağlar



Kimliklendirme Modeli

Kimliklendirmenin başladığı yer AbstractAuthentication ProcessingFilter'dır. Her auth yönteminin kendine özel bir sub class'ı mevcuttur

Başarılı kimliklendirme sonucu Authentication nesnesi oluşur
Farklı auth yöntemlerinin kendine özel token impl mevcuttur



Kimliklendirmeyi AuthenticationManager koordine eder. İşi AuthenticationProvider nesnelerine delege eder

Asıl işlem Auth Provider'da gerçekleşir

Her auth yönteminin kendine özel bir impl mevcuttur

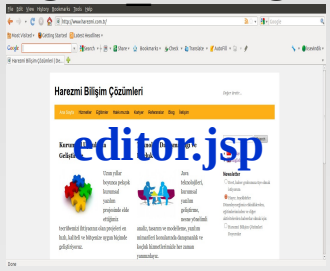
Başarısız Bir Login Akışı

Throws Authentication Exception

1:HTTP request to editor.jsp

FilterSecurity Interceptor

ROLE_EDITOR



2:ExceptionTranlationFilter
Redirect to authentication entry point: login.jsp

3:Submit form to: **/j_spring_security_check**

UsernamePassword Authentication Filter

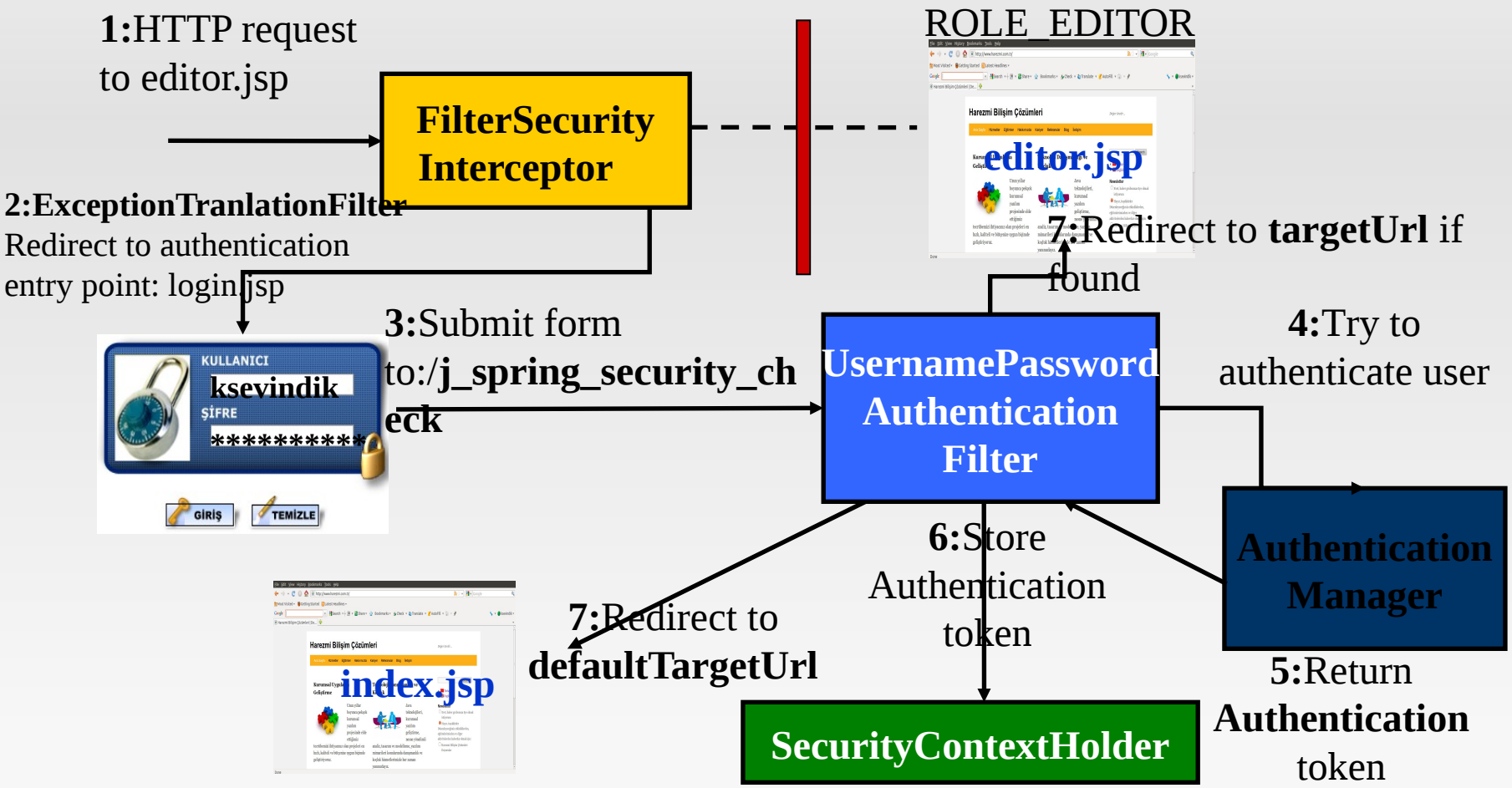
4:try to authenticate user

Authentication Manager

5:throws Authentication Exception

6:Redirect to authentication failure url: login.jsp?hata=true

Başarılı Bir Login Akışı



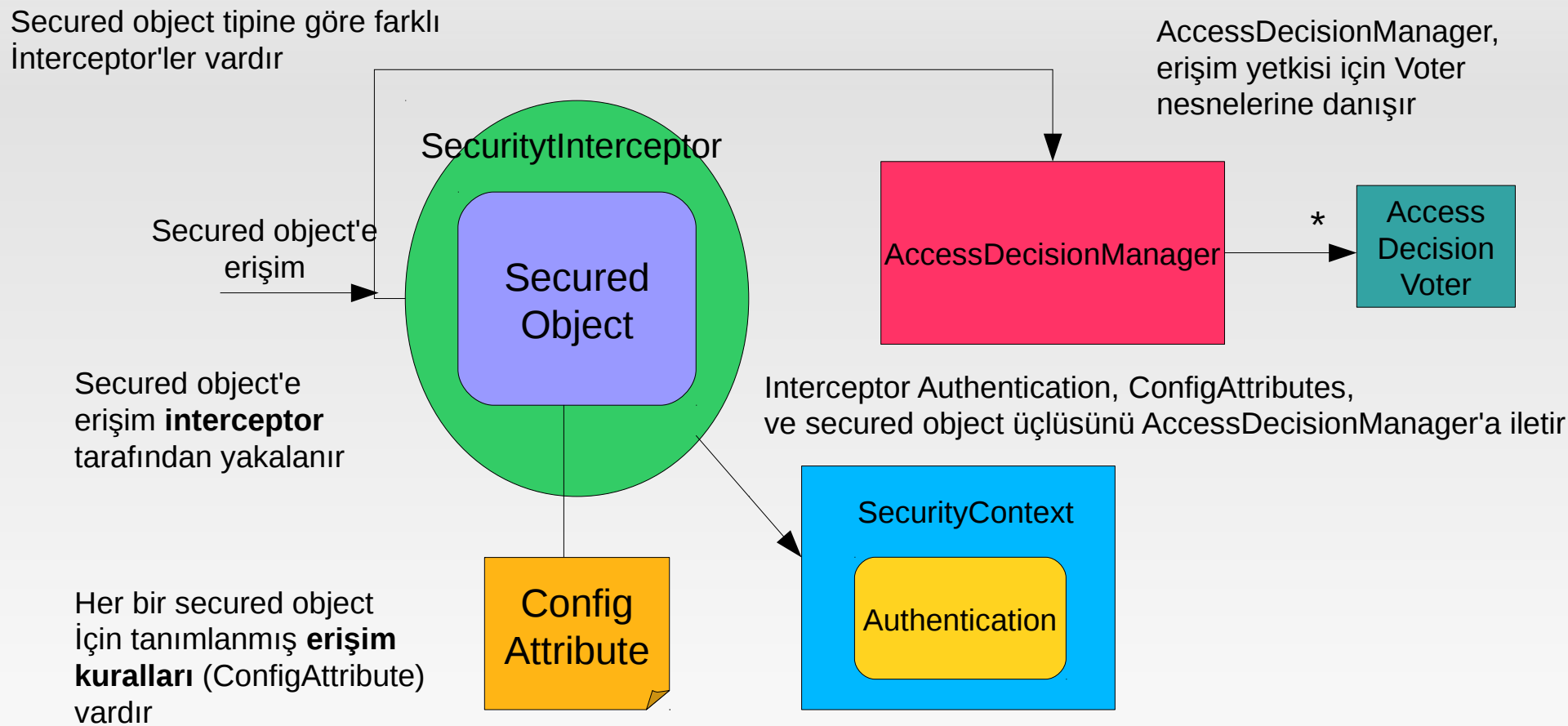
Yetkilendirme

- Yetkilendirme işlemine tabi tutulan herhangi bir nesneye “**secure object**” denir
- Aşağıdakilerden herhangi birisi secure object olabilir
  - Web istekleri
  - Metot çağrıları
  - Domain nesneleri

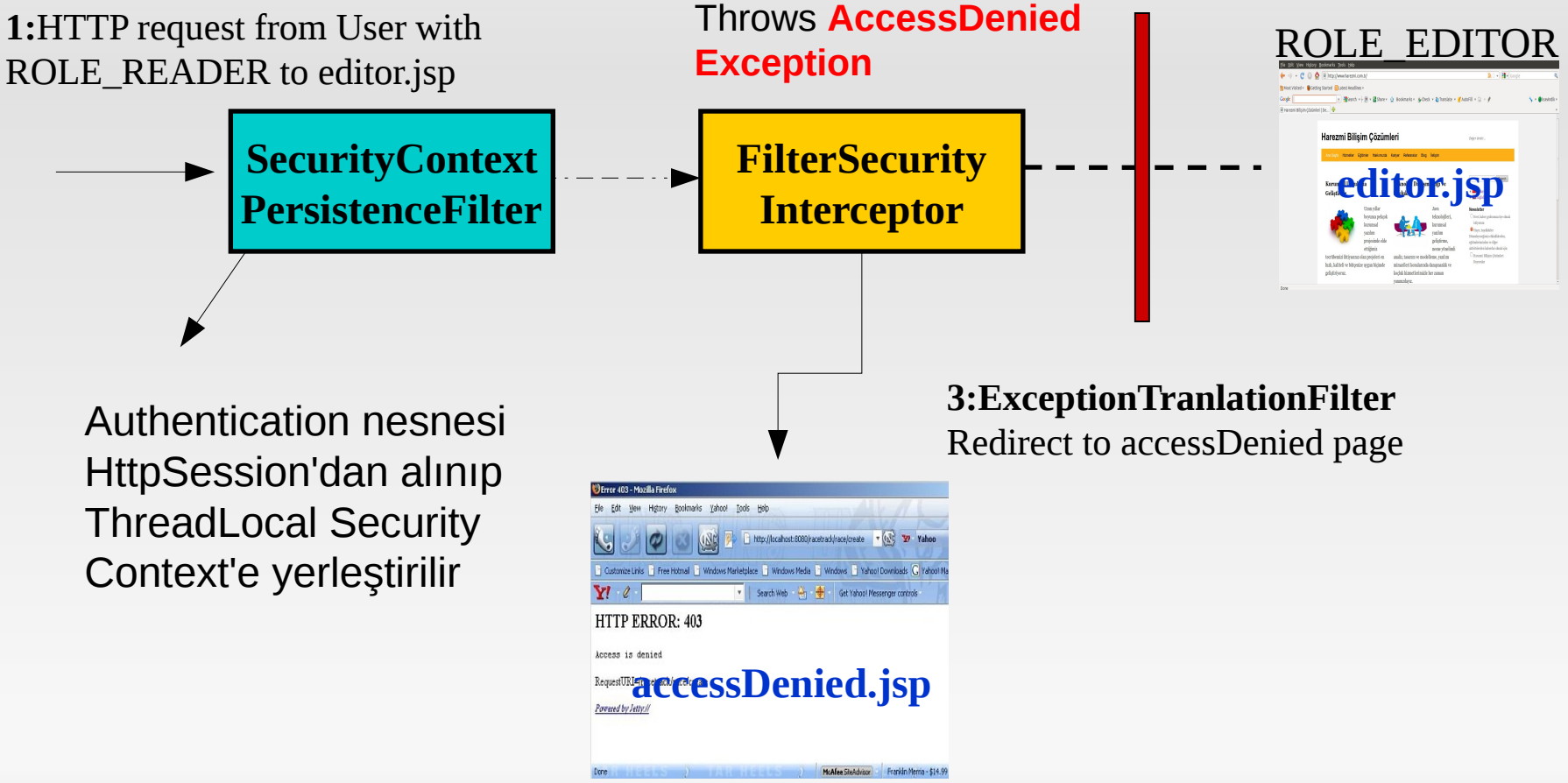
Yetkilendirme

- Spring Security, yetkilendirme için AOP "**around advice**" yöntemini kullanır
- Her bir secure nesne tipine göre **farklı interceptor'ler** devreye girer
  - Web istekleri → FilterSecurityInterceptor
  - Metot çağrıları → MethodSecurityInterceptor
  - Domain nesneleri → AspectJ veya ACL

Yetkilendirmenin İşleyişi



Yetkisiz Erişim Örneği



Namespace Konfigürasyonu

- Acegi Security'deki en büyük problem bean'ların ugun biçimde ve sırada tanımlanması idi
- Spring Security 2.0 ile birlikte gelmiştir
- Basit bir XML tanımı ile bir grup bean konfigürasyonu ile yapılmak istenen tanımlama yapılabilmektedir

Namespace Konfigürasyonu

- **<security:ldap-server/>** tanımı uygulama içinde LDAP ayarlarının yapılması ve test amaçlı embedded LDAP server'ın çalıştırılması için yeterli olmaktadır
- Geliştiriciler LDAP kabiliyetini kullanmak için hangi bean tanımlarının yapılması, hangi property değerlerinin set edilmesi gerektiğini bilmek zorunda değildirler

Namespace ile Spring Security Kullanımı

- Form login konfigürasyonu
- Logout işlemi
- Kriptolu şifrelerin kullanılması
- Beni hatırla kabiliyeti
- Oturum yönetimi
- Web kaynaklarının yetkilendirilmesi
- Metot düzeyinde yetkilendirme

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com