

Tasarım Örüntüleri ile Spring Eğitimi 6

Tasarım Örüntüleri'ne Devam...

High/Low Coupling

High coupling

“Sıkı, iç içe geçmiş ve doğrudan (tightly coupled)” nesne ilişkilerinden uzak dur!

Nesneler arasında ilişkilerin **“gevşek ve dolaylı (loosely coupled)”** olmasına çalış...

Low coupling

Hollywood Prensibi



**Siz bizi aramayın, biz sizi arayacağız!
(Don't call us, we will call you!)**

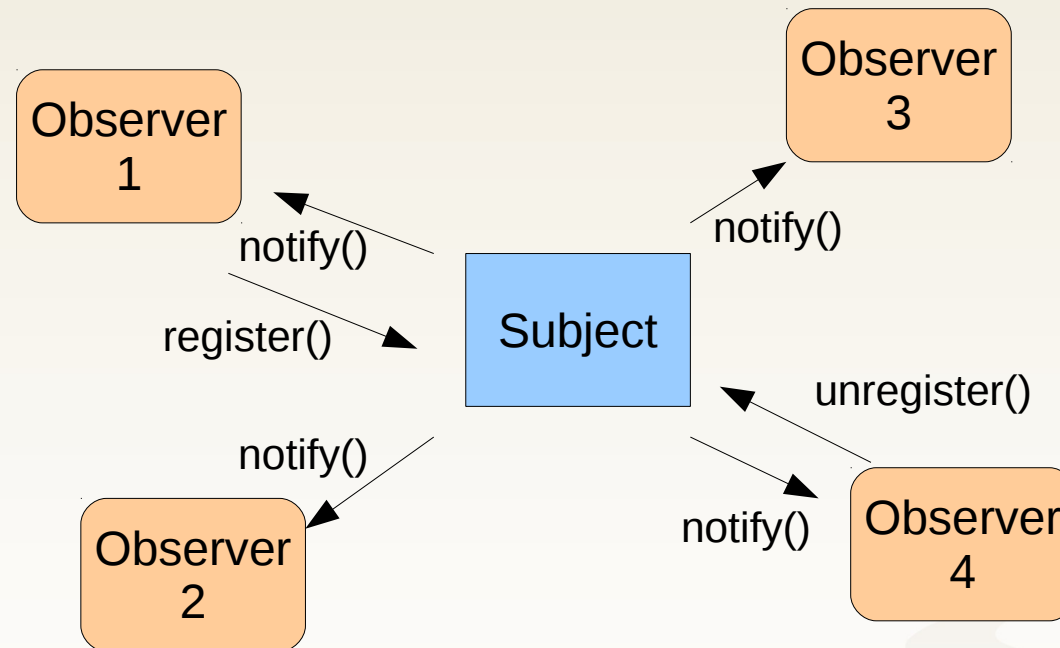
Observer

- Bir grup nesnenin başka **bir nesne(subject)** üzerindeki **değişiklikleri bilmeleri** ve buna göre **işlem yapmaları** gerekebilir
- Bu nesnelerin **subject'i periyodik aralıklarla kontrol etmeleri** bir çözüm olabilir
- Ancak bu **gereksiz bir iletişim trafiği** ortaya çıkaracaktır
- Bunun yerine ilgili nesnelerin **subject'deki değişikliklerden haberdar edilmeleri** daha uygun olacaktır

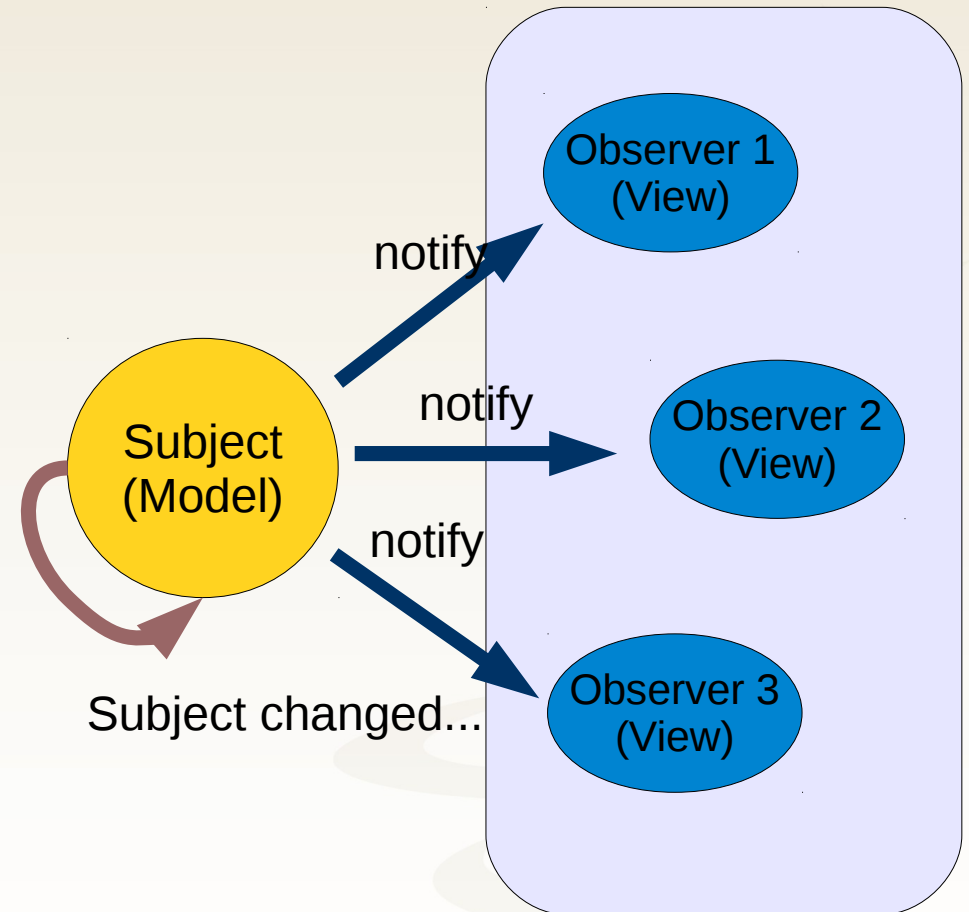
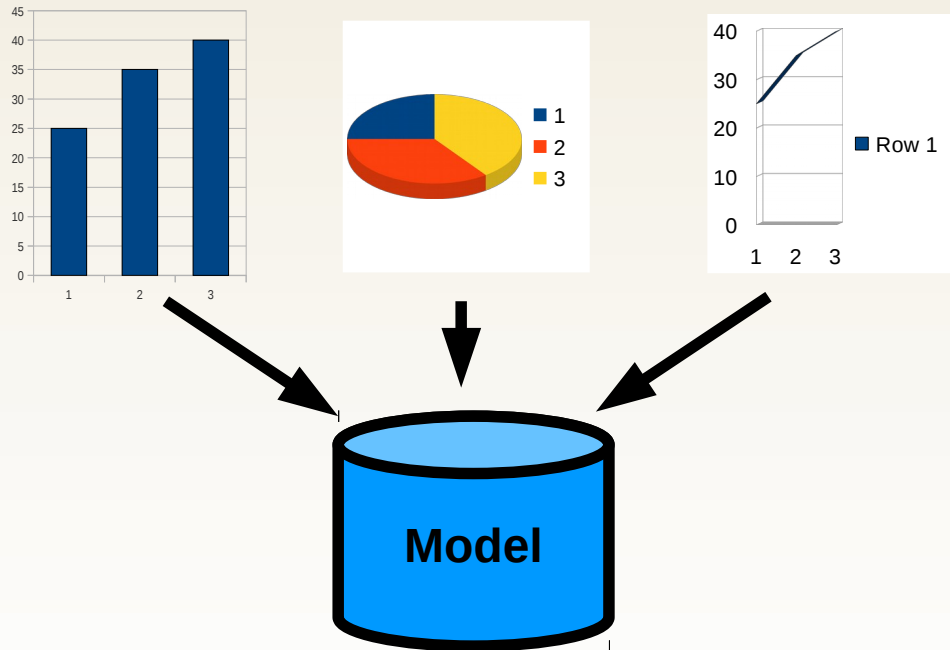
Observer

- Bunun için Observer nesneler kendilerini **subject'e tanıtırlar**
- Subject üzerinde herhangi bir değişiklik olduğunda tanımlı Observer nesneleri **değişiklikten haberdar** edilirler
- Observer nesneler **sadece bilgilendirme (notification) geldiğinde** devreye girerler ve gerekli operasyonları gerçekleştirirler
- Observer'lar subject ile ilgili değişikliklerden haberdar olmak istemedikleri vakit **kendilerini subject'den çıkarmaları** yeterlidir

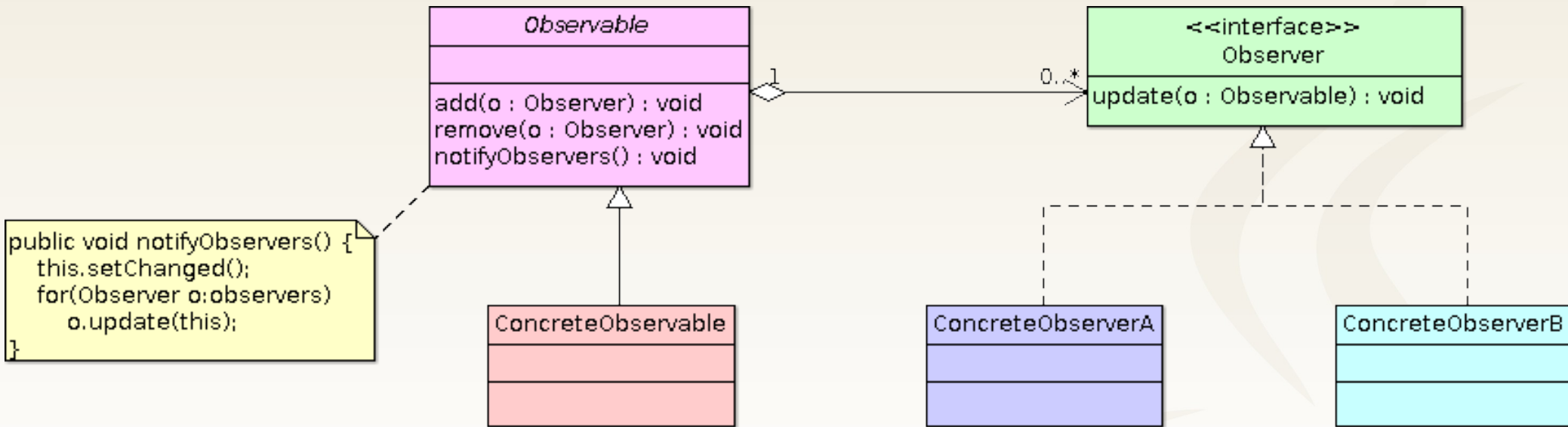
Observer



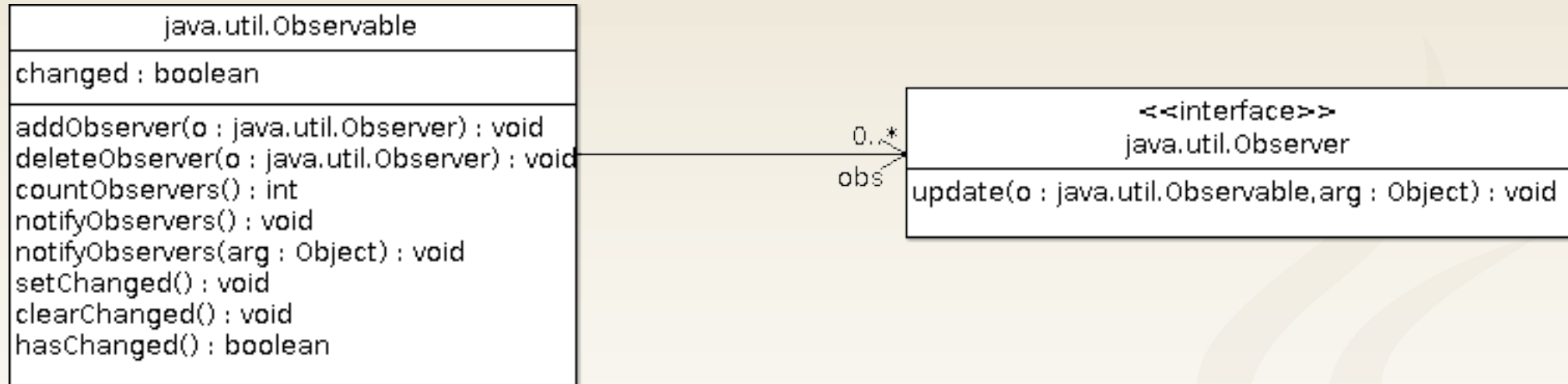
Observer



Observer Sınıf Diagramı

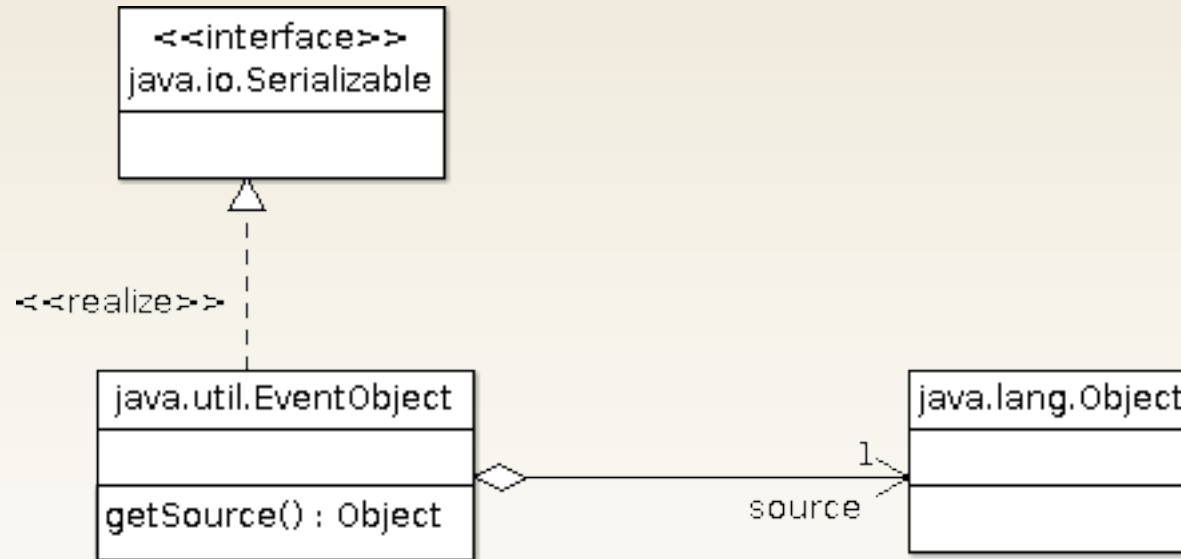


Java ve Observer



JDK içerisinde Observer örüntüsü için sınıflar barındırmaktadır. Uygulama içerisinde Subject kategorisindeki sınıflar Observable sınıfından türeyerek Observer nesnelerini ekleme/çıkarma, kayıtlı Observer nesnelerini kendileri ile ilgili değişikliklerden haberdar etme gibi kabiliyetlere sahip olurlar.

Java ve Observer



Observable nesneler Observer'ları değişikliklerden haberdar etmek için genellikle EventObject'den türeyen nesneleri argüman olarak kullanırlar. EventObject, JDK'daki bütün event state sınıflarının türediği ata sınıftır. uygulamaya özel event state sınıflarının da EventObject sınıfından türetilmeleri önerilir.

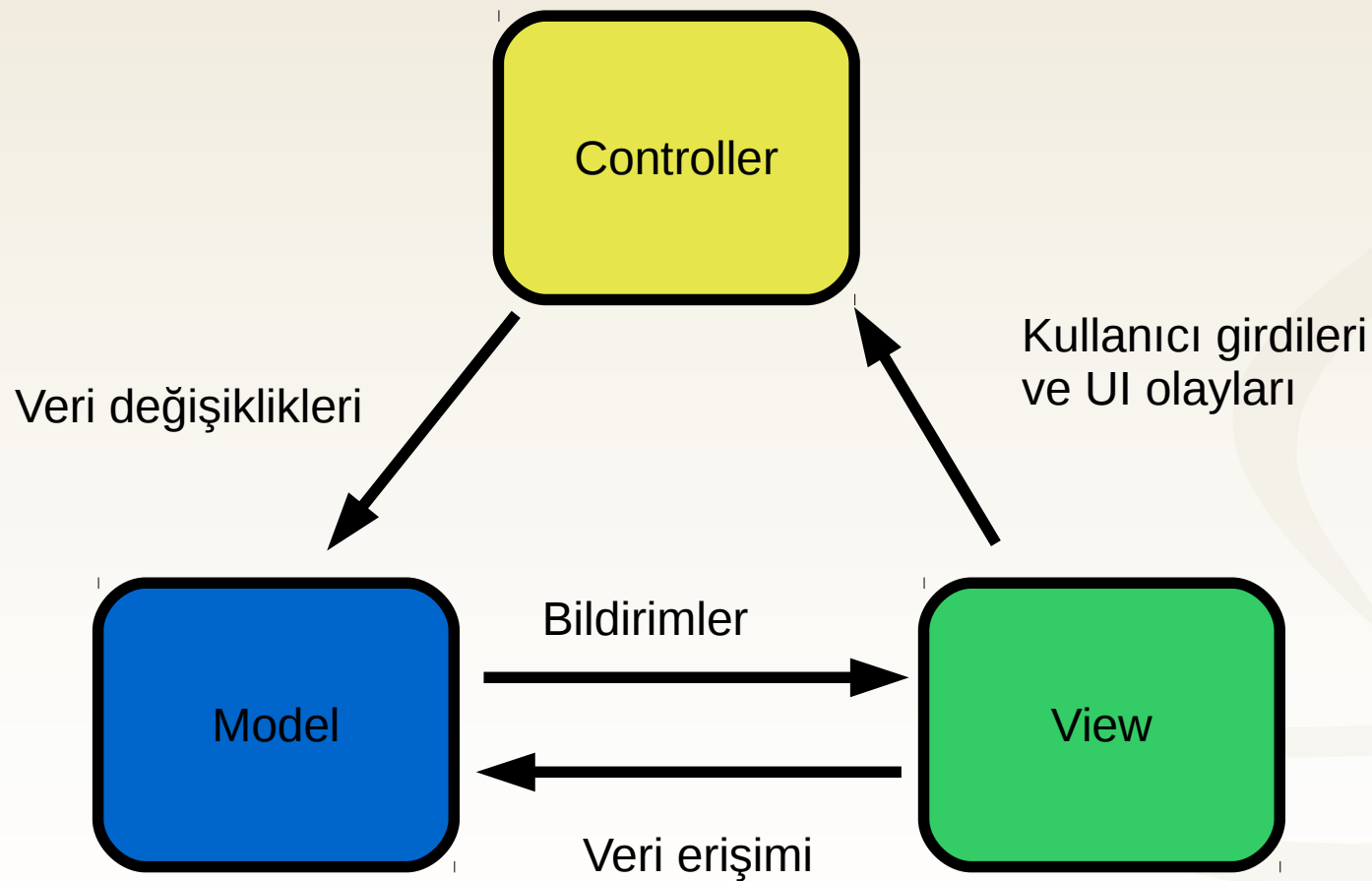
Örüntüsünün Sonuçları

- Subject **hangi tür nesnelerin** kendisi hakkındaki değişikliklerle ilgilendiğini bilmez
- Böylece hem subject'i hem de observer'ları farklı bağlamlarda **birbirlerinden bağımsız** kullanmak mümkün olur
- Bazı durumlarda Observer'lar **subject üzerinde neyin tam olarak değiştiğini** tespit etmekte zorlanabilirler

Model View Controller (MVC)

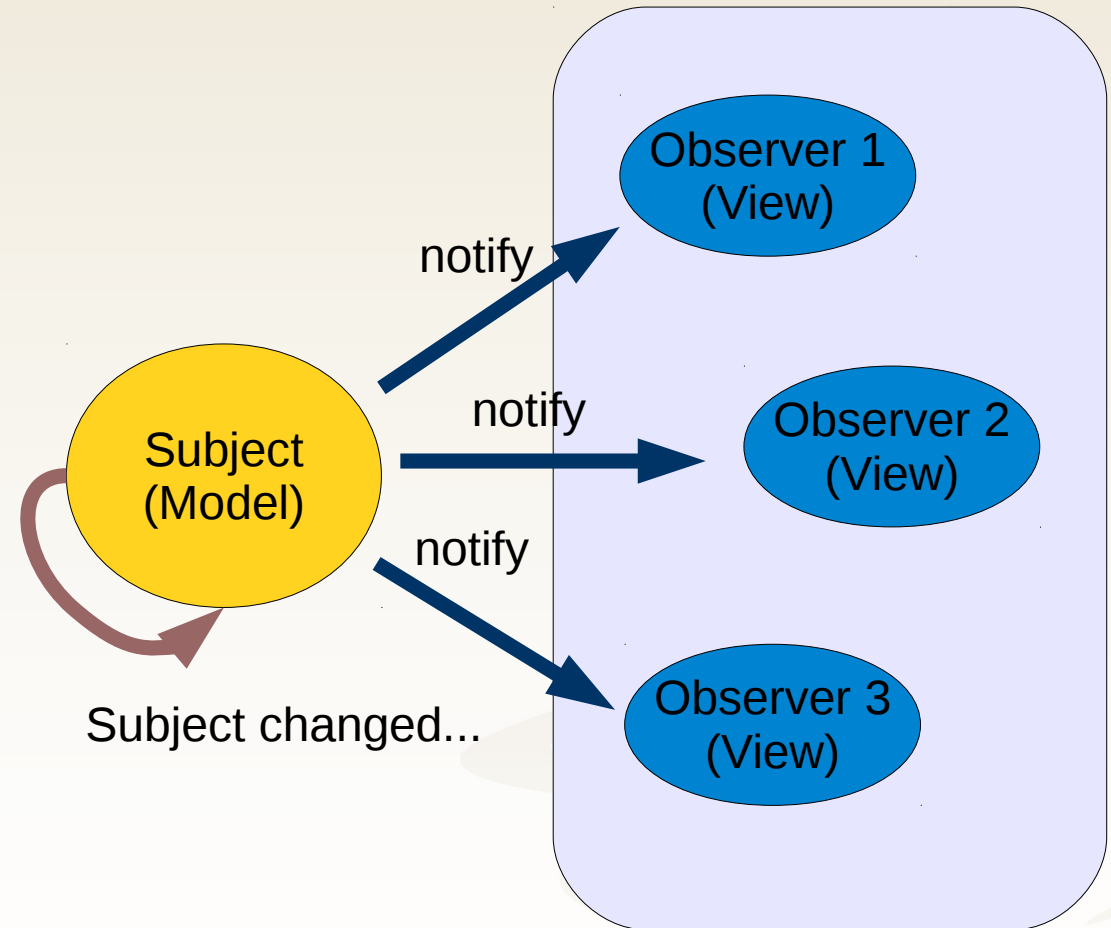
- 70'li yılların sonunda ortaya çıkmıştır
- Norveçli bilim adamı ve mühendis Tyrgve Reenskaug'un Norveç'de başlayıp, Amerika'daki Xerox lablarını ziyareti sırasında gelişmiş **mimarisel bir örüntü**dür
- Kısaca **MVC** olarak da bilinir
- MVC, uygulamanın bileşenlerini **üç ana yapıya** ayırarak ele alır

MVC Bileşenleri Arasındaki Etkileşim



MVC ve Observer

- Model'deki değişiklik **notifikasyonlar** vasıtası ile View(lar) tarafından algılanarak ekrana yansıtılır
- Bu etkileşim **Observer** örüntüsü üzerine kuruludur



MVC'nin Amacı

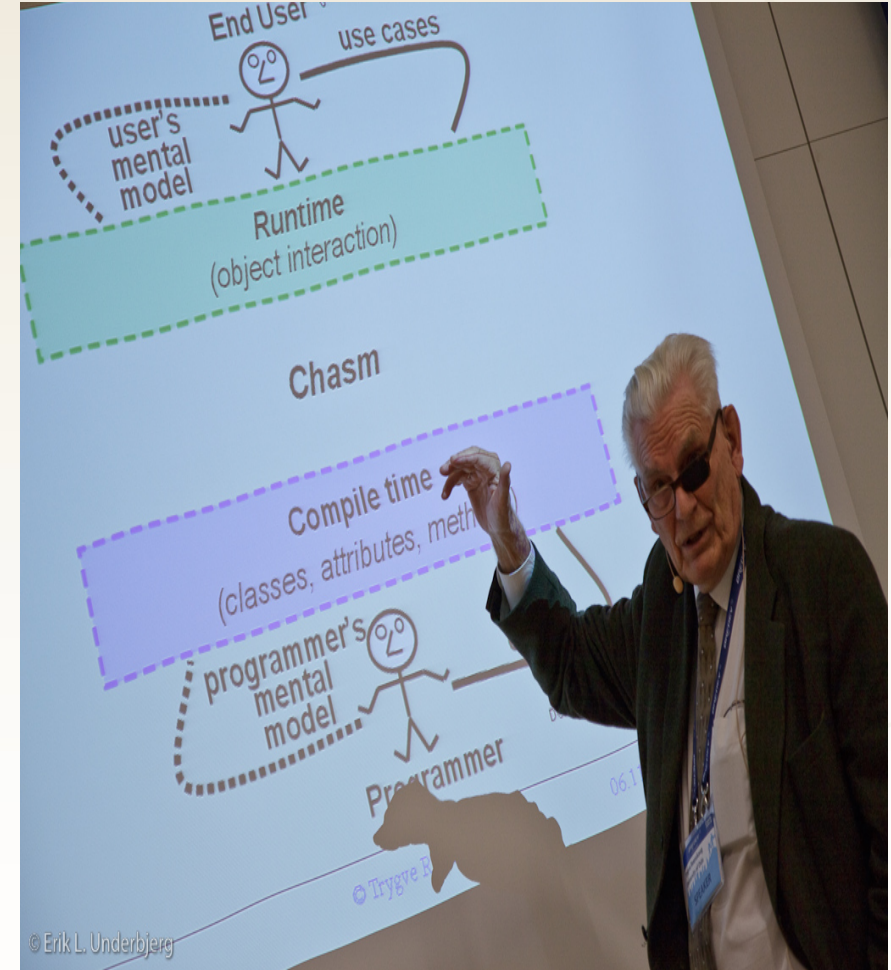
- Güncel pek çok dokümanda MVC'nin amacı olarak “**iş mantığının GUI kodundan ayrılması**” olarak anlatılır
- Her bölümün **kendine özgü** ve diğerlerinden bağımsız bir **sorumluluğu** vardır
- Bu sayede view katmanında herhangi bir değişiklik yapılması gerektiğinde, bunun iş mantığında veya model'de herhangi bir probleme veya değişikliğe yol açmadan kolaylıkla yapılabileceği vurgulanır

MVC'nin Amacı

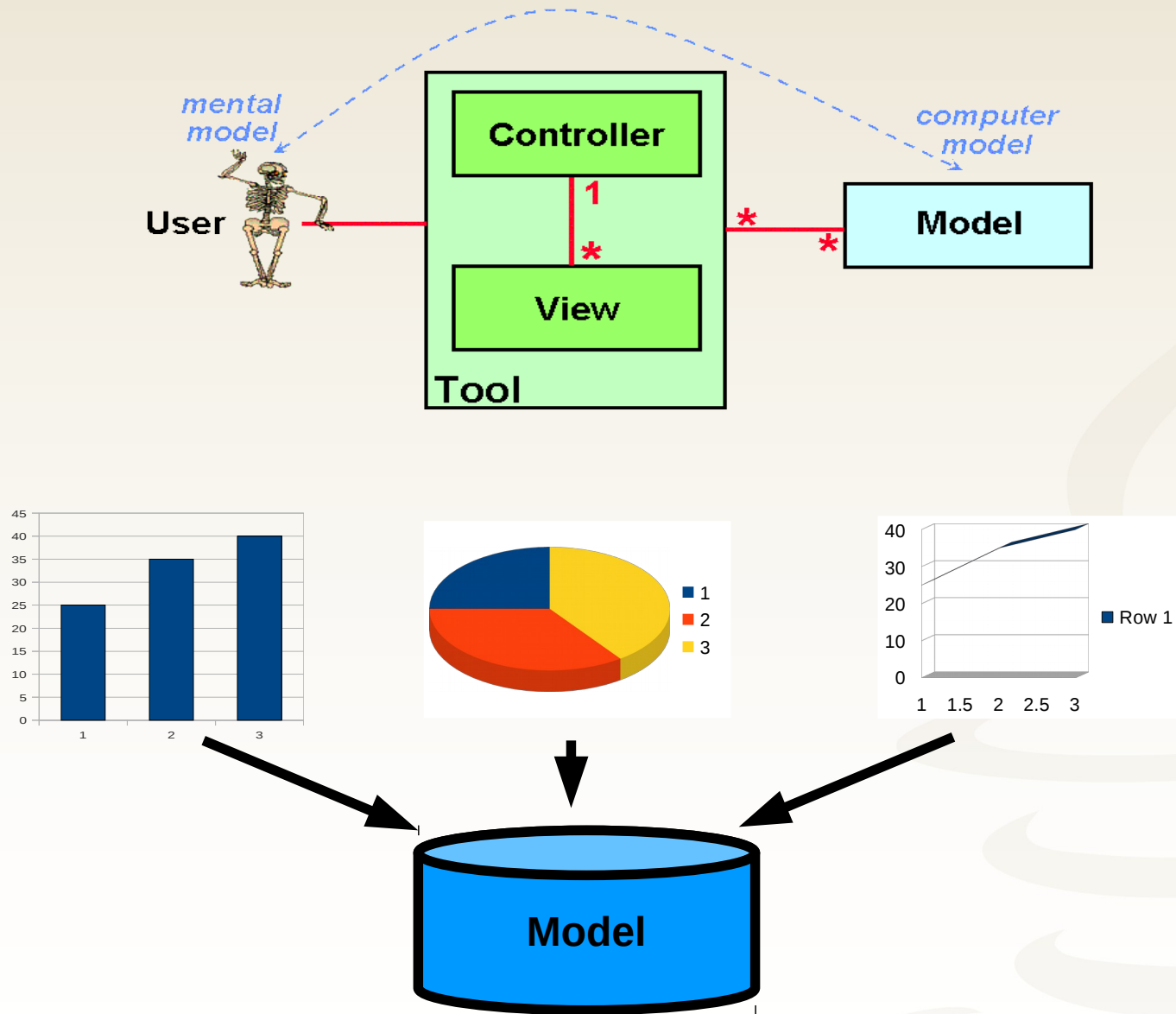
- Oysa MVC'nin mucidi Reenskaug'un asıl amacı daha farklı birşeydi
- MVC'yi anlattığı makalesinde asıl amacın uygulama kullanıcılarının zihinlerindeki **mental model** ile bilgisayar sistemlerindeki **sayısal model** arasındaki **boşluğu dolduran genel bir çözüm** oluşturmak olduğunu belirtmiştir
- Böylece domain verisi (model), doğrudan kullanıcı tarafından erişilebilir, kolaylıkla incelenebilir, yorumlanabilir ve güncellenebilir hale gelecekti

MVC'nin Amacı

- Uygulamayı modüler bir yapıya büründürmek ve **farklı görevleri farklı katmanlara ayıştırmak** MVC için ilk hedef olmamıştı
- MVC makalesinde **“Seperation of Concern”** bir amaç değil sonuçtur



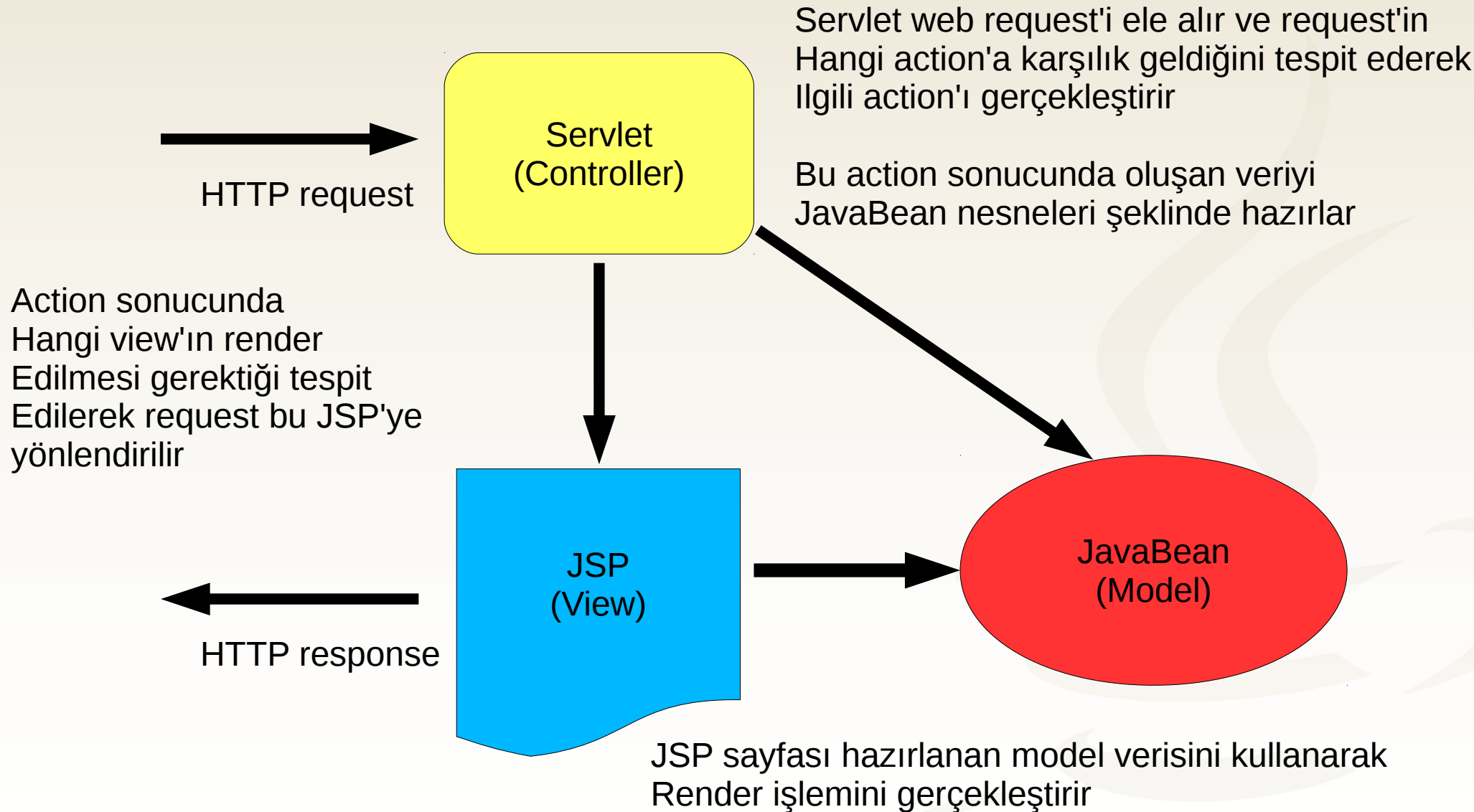
MVC'nin Amacı



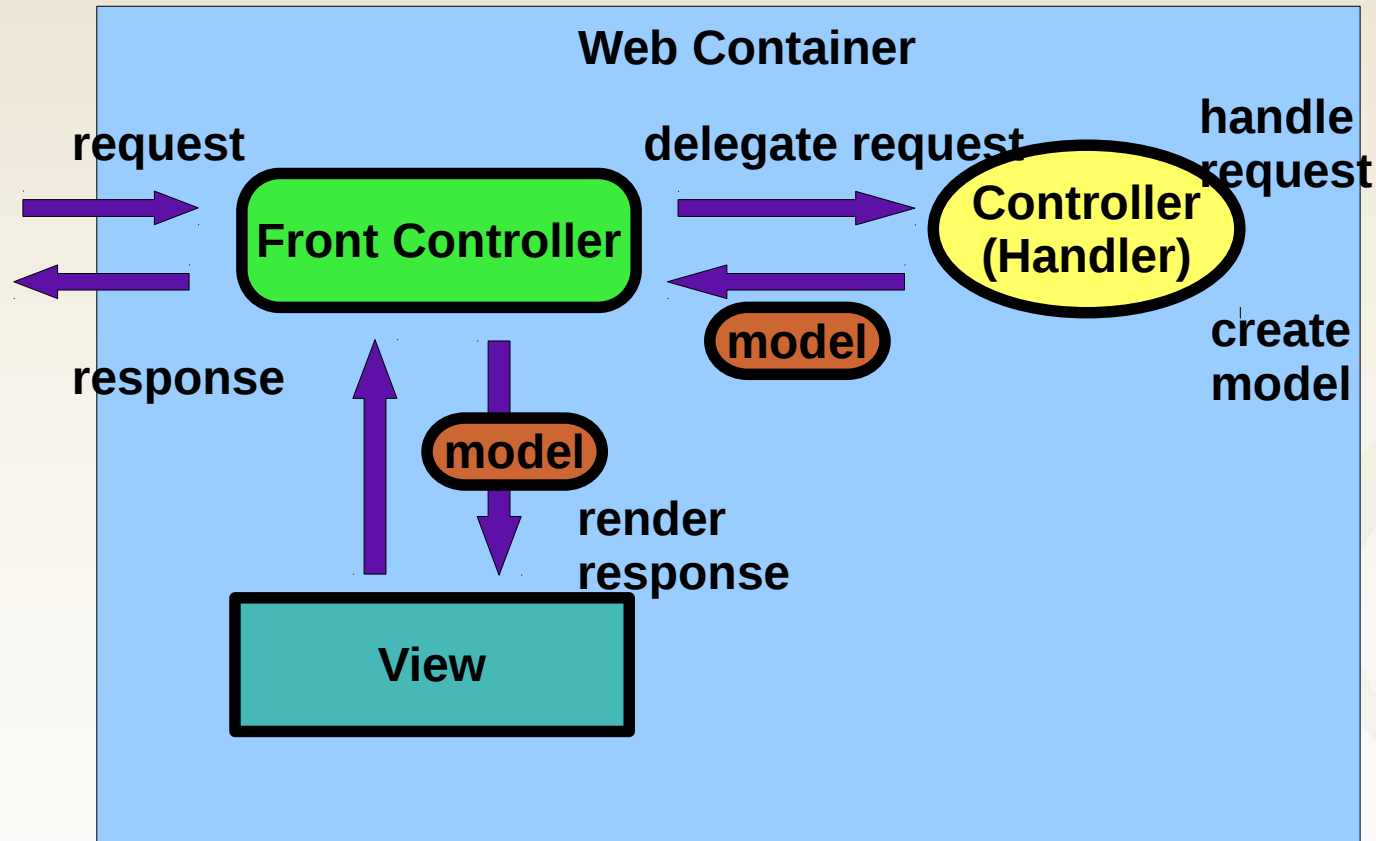
MVC'nin Web'e Uyarlaması: Model2

- Web uygulamalarının gelişim sürecinde Servlet ve JSP'ler ilk çıktığında **view ve iş mantığının iç içe girdiği çözümler** uygulanmakta idi
- Bu çözümler genellikle **Model 1** olarak adlandırılmıştır
- Tam olarak örtüşmese de MVC'yi temel alan mimarisel bir yaklaşım ile **view ve iş mantığı kısımlarını birbirinden ayıran** Web mimarisine ise **Model 2** adı verilmiştir

Model 2 veya MVC2



MVC2 ve Front Controller



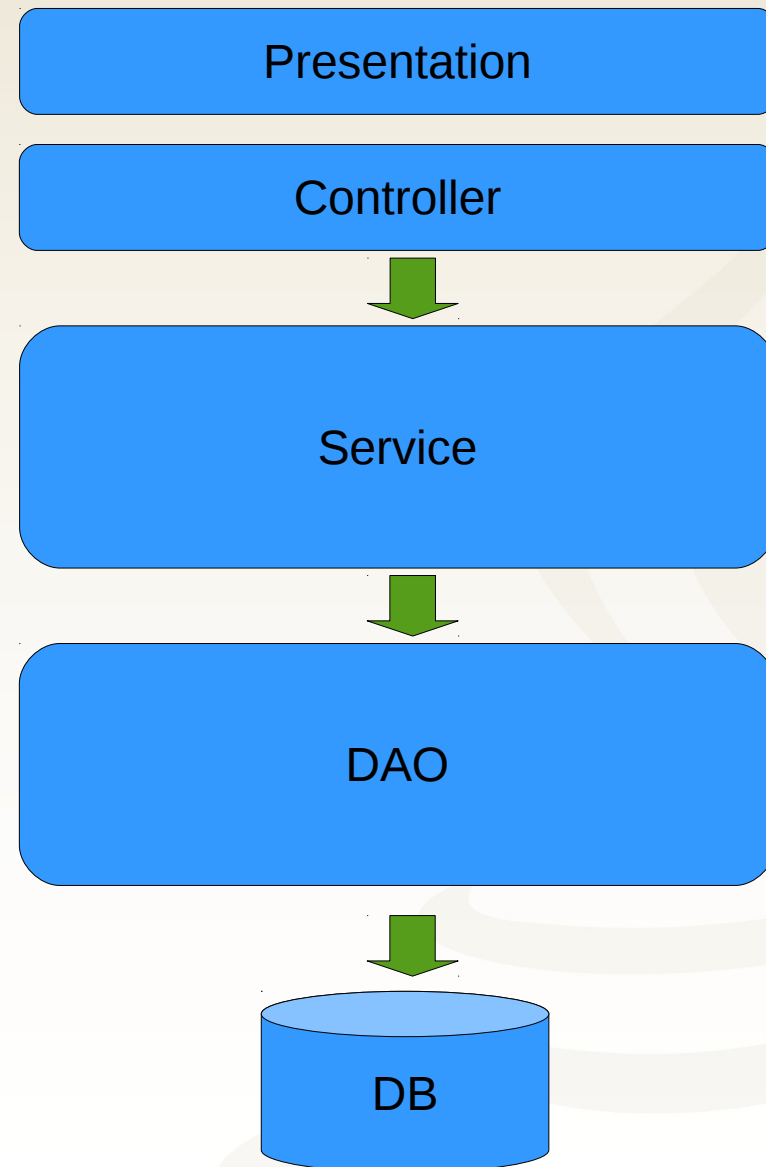
Front Controller genellikle bir Servlet olur, uygulamaya gelen bütün request'leri handle eder

Request'in hangi komuta karşılık geldiği request'in path'inden veya request parametrelerinden anlaşılır

Struts, Spring MVC gibi pek çok action oriented web framework bu örüntü üzerine kurulmuştur. JSF, Vaadin gibi event oriented web framework'lerinin temelinde de bu örüntü yine mevcuttur

MVC ve Katmanlı Mimari

- Java EE ile geliştirilen pek çok uygulamada MVC'nin de etkisi ile **uygulamanın birden fazla katmana ayrılarak geliştirilmesi** yaygın bir mimarisel yaklaşımdır



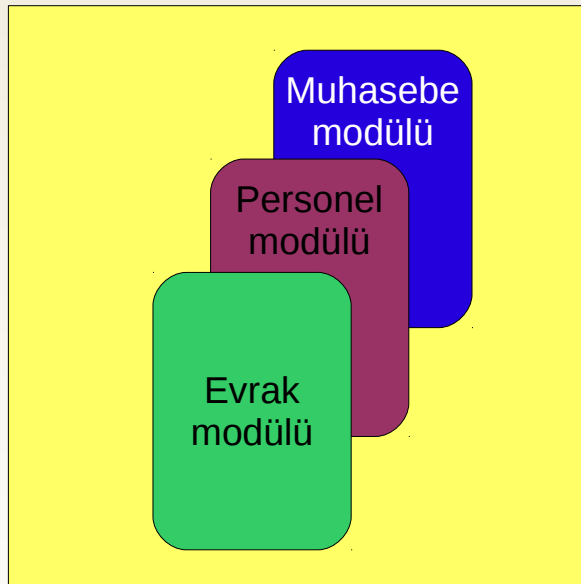
MVC ve Katmanlı Mimari

- Presentation ve Controller katmanları genellikle birlikte **UI katmanı** olarak ele alınırlar
- Kullanıcı isteklerinin işlenmesi, isteklerin servis katmanına iletilmesi ve kullanıcı arayüzünün oluşturulmasından sorumludurlar
- Servis katmanı ise **iş mantığının** yürütülmesinden sorumludur
- Herhangi bir arayüz teknolojisine bağımlı olmadan, **farklı istemcilerden** gelebilecek çağrıları cevaplayabilir

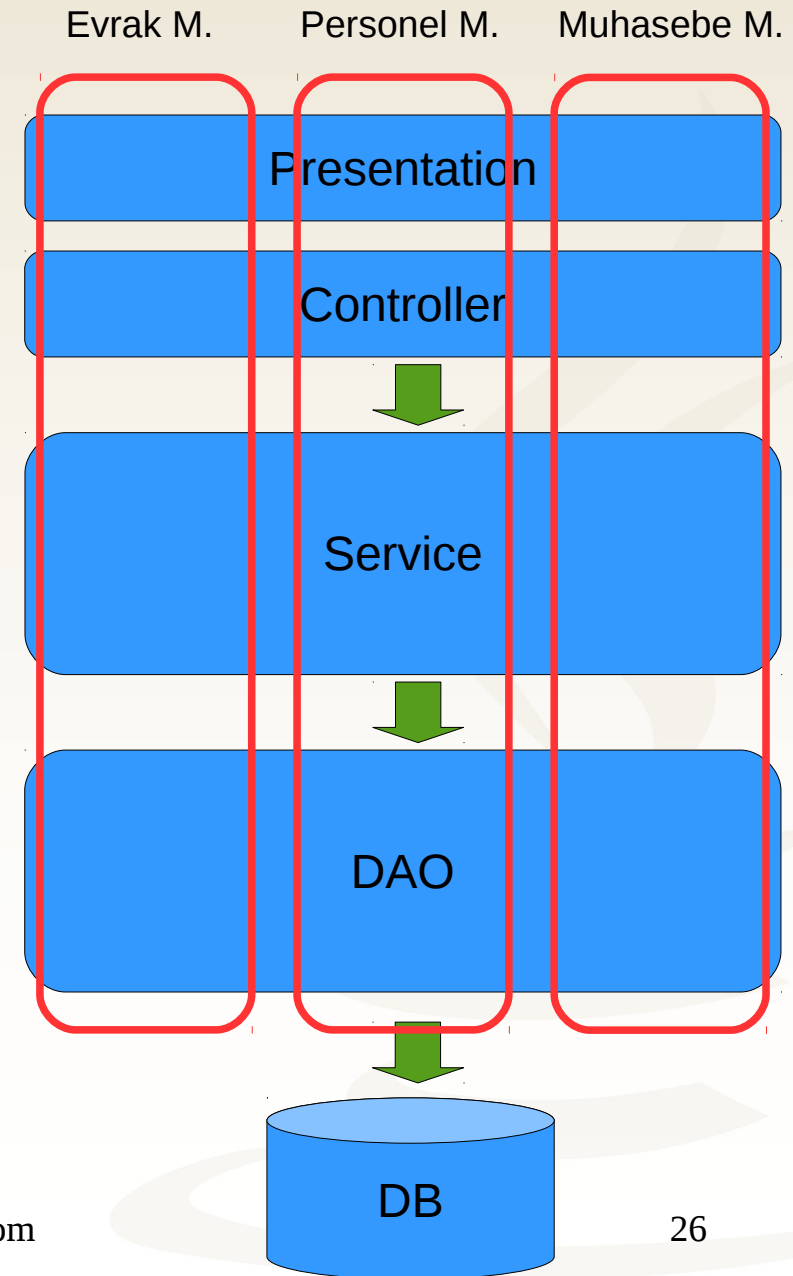
MVC ve Katmanlı Mimari

- Transaction, güvenlik, validasyon gibi ihtiyaçlarda genellikle **servis katmanında** ele alınırlar
- DAO katmanı ise **veri erişiminden** sorumludur
- Kullanılan **persistence teknolojisi** yardımı ile veriye erişim, verinin saklanması, güncellenmesi ve silinmesi gibi ihtiyaçlar bu katman tarafından karşılanır
- Persistence teknolojisinin değişmesine göre bu katmanda değiştirilebilir, **strategy örüntüsüne** karşılık gelir

Katmanlı Mimari ve Modülerlik



Yazılım Sistemi



Web Uygulamalarında Spring Container Konfigürasyonu

Spring ve Web Uygulamaları

- Web uygulamalarında Spring Container genellikle **web.xml'de deklaratif biçimde** tanımlanarak yaratılır
- Spring **herhangi bir Web UI teknolojisi** ile birlikte kullanılabilir
- Web uygulamalarındaki Spring Container'ın tipi **WebApplicationContext**'tir
- **WebApplicationContext** **ApplicationContext'i extend** eder

Web Uygulamalarında ApplicationContext Oluşturma

- Web ortamı için **ilave kabiliyetler** sunar
- `WebApplicationContext`
ContextLoaderListener ile bootstrap sırasında yaratılır
- Yaratılan `WebApplicationContext` daha sonra **ServletContext** üzerinden erişilebilir
- Örneğin bir Servlet içerisinde `ApplicationContext` **bean lookup** ile istenilen bean alınıp kullanılabilir

Web Uygulamalarında ApplicationContext Oluşturma

web.xml

```
<web-app>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath*:/appcontext/beans-*.xml
    </param-value>
  </context-param>
```

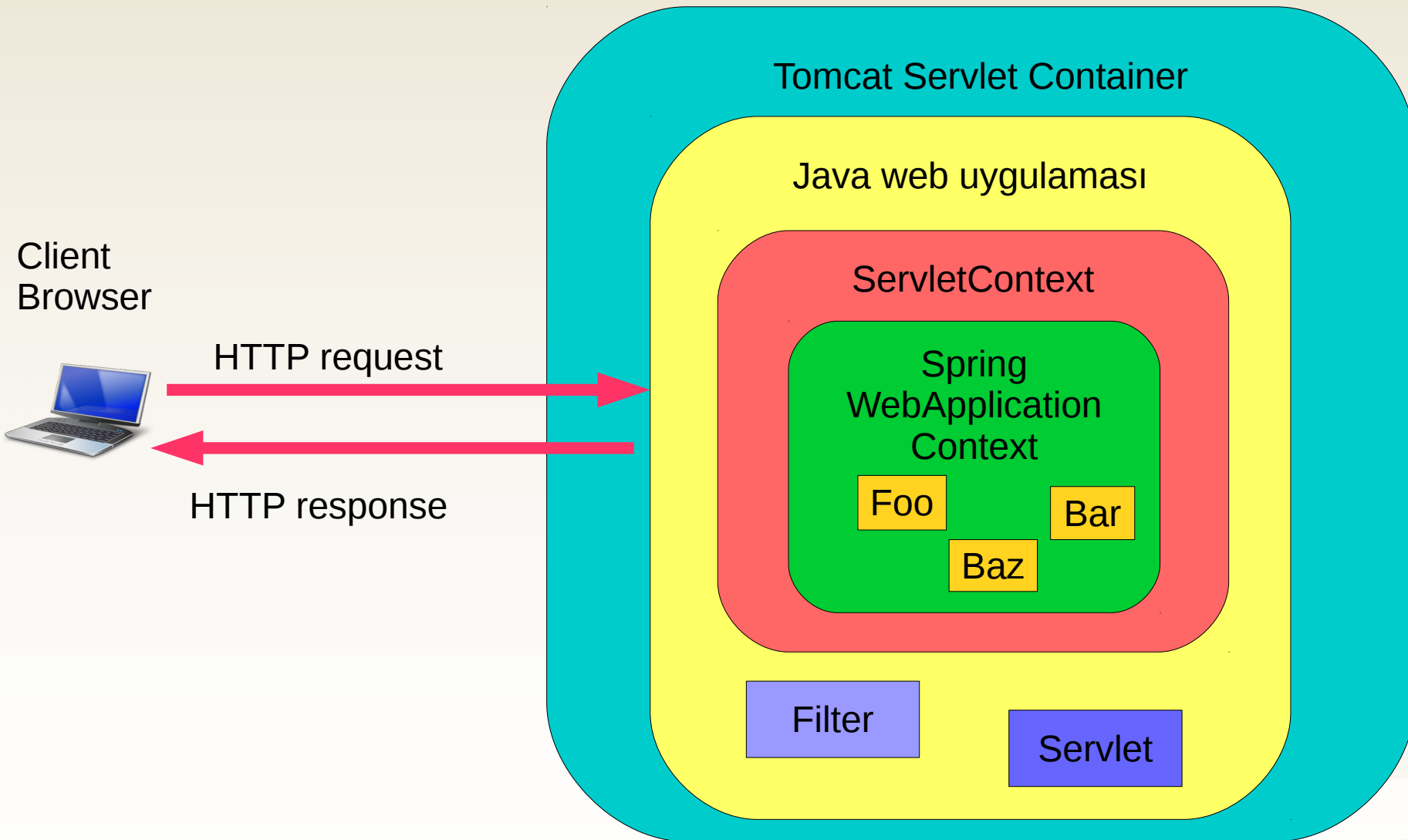
- Default: /WEB-INF/applicationContext.xml
- Eğer parametre mevcut ise birden fazla dosya path'i belirtilebilir
- Path'ler virgül, noktalı virgül veya boşluk ile ayrılabilir
- Ant-style path pattern'ları da desteklenir

```
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
</web-app>
```

WebApplicationContext'e Nasıl Erişilir?

- ServletContext'e bind edilen WebApplicationContext'e erişmek için Spring bir **utility sınıf** sunar
- **WebApplicationContextUtils** sınıfı ile Servlet vb nesneler içerisinde WebApplicationContext'e erişilebilir
 - `getRequiredWebApplicationContext(servletContext)`
 - `getWebApplicationContext(servletContext)`

ServletContext & WebApplicationContext İlişkisi

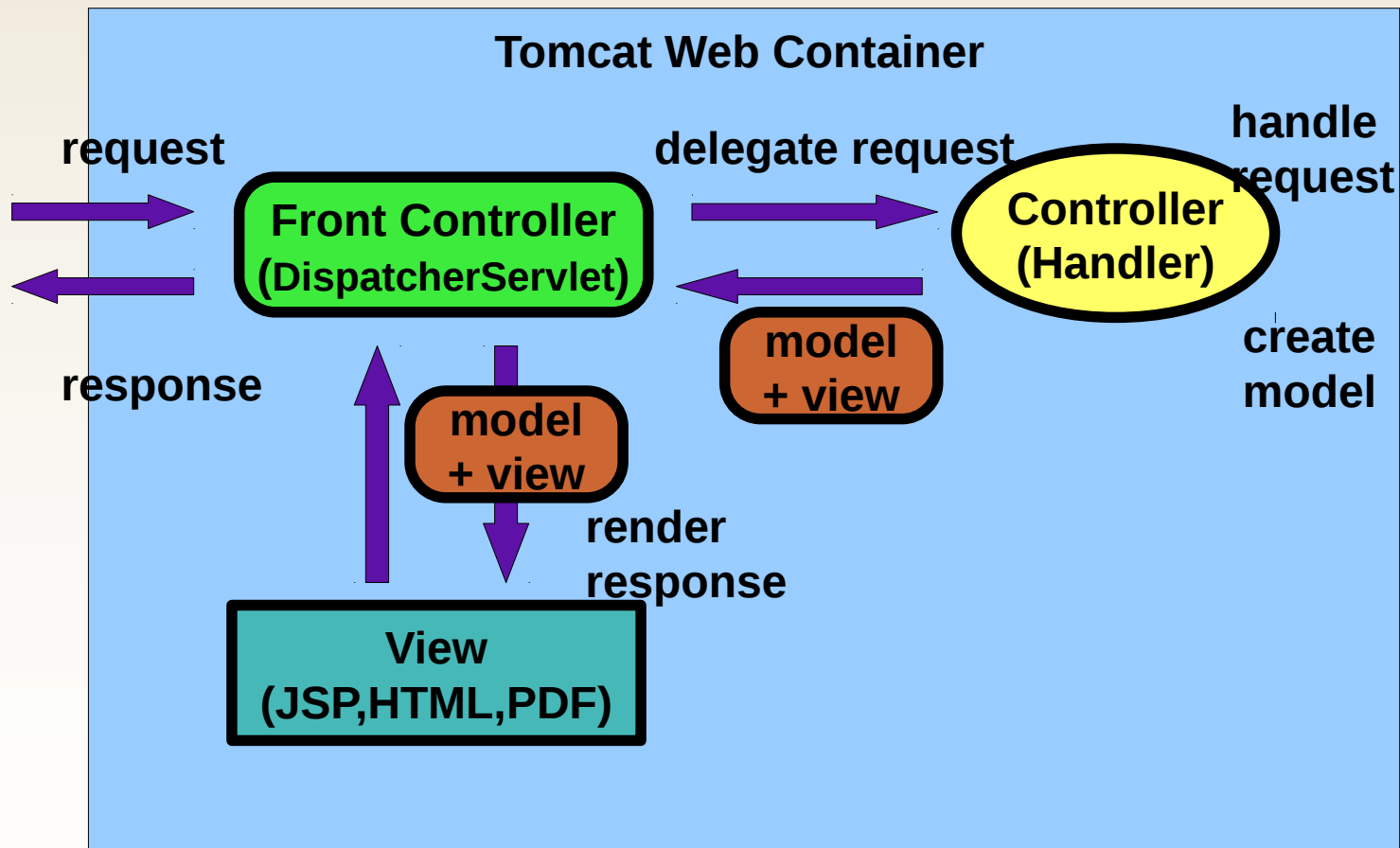


Spring Web MVC

Spring MVC ve DispatcherServlet

- Spring MVC de diğer pek çok MVC framework gibi **request ve action tabanlı** bir frameworktür
- **DispatcherServlet** etrafında kuruludur
- Bu servlet, Spring IoC Container ile **tam bir entegrasyona** sahiptir
- DispatcherServlet, **Front Controller** örüntüsünü implement eder

Front Controller & DispatcherServlet



DispatcherServlet Konfigürasyonu

- DispatcherServlet'in aktivasyonu için **web.xml** içerisinde tanım yapılarak, burada **hangi requestleri ele alacağı** belirtilir

web.xml

```
<web-app>
  <servlet>
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-class>
org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>/mvc/*</url-pattern>
  </servlet-mapping>
</web-app>
```

DispatcherServlet Konfigürasyonu

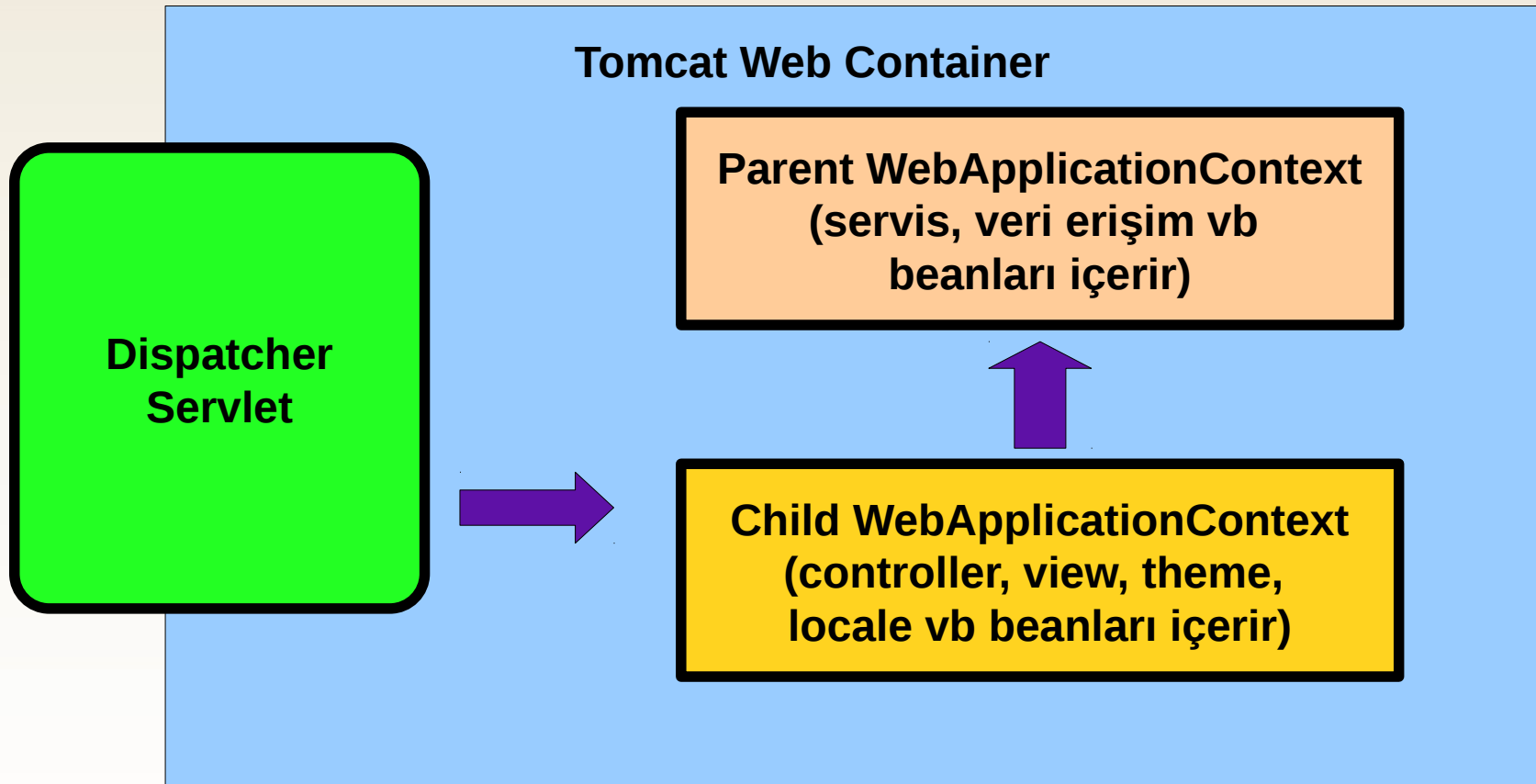
- Default olarak **WEB-INF** dizini altında **[servlet-name]-servlet.xml** isminde bir Spring bean konfigürasyon dosyası arar
- Bu dosyayı yükleyerek **kendine ait bir WebApplicationContext** yaratır
- Bu WebApplicationContext, eğer tanımlı ise **ContextLoaderListener** ile yaratılan **WebApplicationContext'in çocuğudur**

DispatcherServlet Konfigürasyonu

- WEB-INF dizini altındaki dosyanın lokasyonu veya adı istenirse **contextConfigLocation** servlet init parametresi ile değiştirilebilir, ya da **birden fazla dosya** lokasyonu belirtilebilir

```
<servlet>
  <servlet-name>DispatcherServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:/appcontext/mvc-servlet.xml</param-value>
  </init-param>
</servlet>
```

DispatcherServlet ve ApplicationContext Hiyerarşisi



DispatcherServlet ve ApplicationContext Hiyerarşisi

- DispatcherServlet'a ait çocuk `WebApplicationContext` içerisinde, **`ContextLoaderListener`** ile yaratılan **root `WebApplicationContext`** içerisindeki bütün bean'lere erişilebilir
- Ancak parent `ApplicationContext` içerisinde **çocuk `ApplicationContext`'deki** bean'lara erişim mümkün değildir

DispatcherServlet ve ApplicationContext Hiyerarşisi

- Eğer DispatcherServlet'ın kendine ait child WebApplicationContext'inin olması istenmiyor ise **contextConfigLocation** init parametre değeri boş bırakılmalıdır

```
<servlet>
  <servlet-name>DispatcherServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value></param-value>
  </init-param>
</servlet>
```

Spring MVC Özelliklerinin Aktivasyonu

- `<mvc:annotation-driven />` elemanı ise aşağıdaki kabiliyetleri devreye sokar:
 - Built-in **HttpMessageConverter** nesnelerini register eder
 - **Formatter** ve **Conversion** servislerini devreye alır
 - Controller handler metot parametrelerindeki **@NumberFormat**, **@DateTimeFormat** anotasyonları devreye sokar

Spring MVC Özelliklerinin Aktivasyonu

- **@ControllerAdvice**'in devreye girmesini sağlar
- Controller metotlarında **validasyon kabiliyetini** devreye sokar
- Controller handler metotlarının input argümanlarında veya return değerinde **@Valid** anotasyonu kullanılabilir

Controller Bean'ları

- Web katmanından **servis katmanına erişim** sağlayan bean'lardır
- HTTP **request**'lerini **handle** ederler
- Kullanıcı **input'unu alır ve modele dönüştürür**
- Spring 3 ile birlikte Controller beanları için herhangi bir **arayüz implement edilmesi, sınıftan türetilmesi** gerekmez
- Sınıf düzeyinde **@Controller** anotasyonu ile tanımlanırlar

Controller Bean'ları

- Controller bean'ları **DispatcherServlet'in kendi WebApplicationContext'i** içerisinde tanımlanmalıdır
- Bu XML tabanlı biçimde yapılabilir, yada **`<context:component-scan />`** elemanı kullanılabilir

@RequestMapping Annotasyonu

- HTTP request'leri ile **@Controller** bean'lerinin **@RequestMapping** annotasyonuna sahip metotları eşleştirilir
- Bu metotlara **handler metot** adı verilir
- Bir controller sınıfında **birden fazla** handler metot yer alabilir
- Handler metotlarının **alabileceği input parametreleri ve return değerinin** ne olabileceği oldukça esnektir

@RequestMapping Annotasyonu

@Controller

public class HelloWorldController {

@RequestMapping("/hello")

public ModelAndView helloWorld() {

ModelAndView mav = new ModelAndView();
mav.setViewName("/hello.jsp");
mav.addObject("message", "Hello World!");

return mav;

}

}

URL requestinin controller metotları ile eşleştirilmesini sağlar

Controller handler metodunun hem model, hem de view bilgisini tek bir return değeri olarak dönmesini sağlar

Sınıf Düzeyinde @RequestMapping Annotasyonu

- @RequestMapping anotasyonu **sınıf veya metot düzeyinde** kullanılabilir
- Sınıf düzeyinde **opsiyoneldir**
- Kullanıldığında sınıf düzeyindeki değer metotlardaki @RequestMapping tanımlarına **prefix** olarak eklenir

Sınıf Düzeyinde @RequestMapping Annotasyonu

Sınıf düzeyinde tanımlandığı takdirde metod düzeyindeki tanımlar relatif hale gelir

```
@Controller  
@RequestMapping("/greet")  
public class HelloWorldController {
```

```
    @RequestMapping("/hello")  
    public ModelAndView hello() {  
        // ...  
    }
```

hello() metodu eğer
URI /greet/hello şeklinde
olursa çağrılacaktır

```
    @RequestMapping("/bye")  
    public ModelAndView bye() {  
        // ...  
    }  
}
```

bye() metodu eğer
URI /greet/bye şeklinde
olursa çağrılacaktır

Controller Metot

Return Tipi: String

@Controller

```
public class HelloWorldController {
```

```
    @RequestMapping("/hello")
```

```
    public String helloWorld(ModelMap model) {  
        model.addAttribute("message", "Hello World!");  
        return "/hello.jsp";  
    }
```

```
}
```

```
}
```



Controller metodunun return tipi String ise, bu “**logical view**” ismine karşılık gelir

Controller Metot

Return Tipi: void

@Controller

```
public class HelloWorldController {
```

```
    @RequestMapping("/hello")
```

```
    public void helloWorld(HttpServletRequest response) {
```

```
        response.getWriter().write("Hello World!");
```

```
    }
```

```
}
```



Return tipi void ise, bu DispatcherServlet'e “**response'u controller metodu üretecek, sen herhangi bir şey yapma!**” demektir

@ResponseBody Annotasyonunun İşlevi

```
@RequestMapping("/hello")  
@ResponseBody  
public String helloWorld() {  
    return "Hello World";  
}
```



Metot return değeri http response body'sini oluşturur

@RequestBody

Annotasyonunun İşlevi

```
@RequestMapping(value="/printRequestBody",method=RequestMethod.POST)
public void handleRequest(@RequestBody String body, Writer writer)
throws IOException {
    writer.write(body);
}
```



Http request body'si @RequestBody ile işaretlenen metot parametresine atanır

@RequestParam

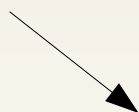
Request parametresinin değerini
metot parametresine atar

```
@RequestMapping("/pet")
public String displayPet(@RequestParam("petId") int petId,
    ModelMap model) {
    Pet pet = this.clinic.loadPet(petId);
    model.addAttribute("pet", pet);
    return "petForm";
}
```

<http://localhost:8080/petclinic/pet?petId=123>

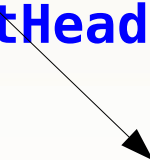
@CookieValue ve @RequestHeader

```
@RequestMapping("/displaySessionId")  
public void displaySessionId(  
    @CookieValue("JSESSIONID") String cookie) {  
    //...  
}
```



Http cookie değerini metod parametresine bind eder

```
@RequestMapping("/displayHeaderInfo")  
public void displayHeaderInfo(  
    @RequestHeader("Accept-Encoding") String encoding,  
    @RequestHeader("Keep-Alive") long keepAlive) {  
    //...  
}
```



Http request header değerini metod parametresine bind eder

Controller Bean'ları ve Exception'lar

- Handler metotlarda meydana gelen exception'ları yakalamak için **exception handler metotlar** tanımlanabilir
- Metot üzerine **@ExceptionHandler** anotasyonu ile tanımlanır
- Her controller bean'ı için **ayrı ayrı** tanımlanmalıdır

Controller Bean'ları ve Exception'lar

@Controller

```
public class HelloController {
```

```
    @RequestMapping("/hello")
```

```
    public String helloWorld(ModelMap model) {
```

```
        model.addAttribute("message", "Hello World!");
```

```
        if(true) throw new RuntimeException("error!!!");
```

```
        return "/hello.jsp";
```

```
    }
```

```
    @ExceptionHandler(RuntimeException.class)
```

```
    public void handle(RuntimeException ex, Writer writer) {
```

```
        writer.write("Error handled : " + ex);
```

```
    }
```

```
}
```

Birden fazla exception tipi alabilir
Değer olarak normal handler metotlar gibi view dönebilir,
yada response'u kendisi üretebilir

@ControllerAdvice ve Global Exception Handling

- @RequestMapping ile işaretlenmiş ve farklı Controller sınıflarındaki handler metotlar için geçerli olacak bir takım **yardımcı metotların tek bir sınıfta toplanmasını** sağlar
- Bu yardımcı metotlar **@ExceptionHandler**, **@InitBinder**, **@ModelAttribute** gibi anotasyonlarla tanımlanmaktadırlar

@ControllerAdvice ve @ExceptionHandler

```
@ControllerAdvice
public class GlobalErrorHandler {

    @ExceptionHandler(IOException.class)
    public String handleException(IOException ex,
    HttpServletRequest request) {
        request.setAttribute("exception", ex);
        return "/error.jsp";
    }
}
```

@ControllerAdvice ve Paket Düzeyinde Sınırlandırma

- Normalde @ControllerAdvice ile işaretlenen **bean bütün Controller bean'ları** için geçerlidir
- Ancak ControllerAdvice'ın **hangi Controller bean'ları için geçerli olacağı** da belirtilebilir

```
@ControllerAdvice(basePackages={"com.javaegitimleri", "tr.com.harezmi"})  
public class ErrorHandler {  
    ...  
}
```

↙
Filtreleme anotasyon veya tiplere göre de yapılabilir

URI Template Kabiliyeti

URL içerisindeki bölümlere controller metodu içerisinde erişmeyi sağlar, her bir bölüm bir değişkene karşılık gelir. Bu değişkenlerin requestdeki karşılıkları controller metod parametrelerine aktarılır.

```
@RequestMapping("/{ownerId}")
public String findOwner(@PathVariable("ownerId") Long
id, Model model) {
    // ...
}
```

`http://localhost/owners/1` --> `ownerId=1`

@PathVariable ile değişken değeri metod parametresine aktarılır

Metod parametresi herhangi bir basit tip olabilir: **int, long, String**

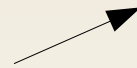
URI Template Kabiliyeti

URI template variable isimleri ile metot parametre isimlerinin eşleştirilmesi **derleme işleminin debug özelliği** açıksa mümkün olur. Aksi takdirde @PathVariable'a **variable ismi** verilmelidir

```
@RequestMapping("/{ownerId}")  
public String findOwner(@PathVariable Long ownerId, Model  
model) {  
    // ...  
}
```

URI Template Kabiliyeti ve Wildcard Kullanımı

Ant stili örüntüleri de destekler



```
@RequestMapping("/owners/*/pets/{petId}")  
public String findPet(@PathVariable Long petId, Model model) {  
  
    Pet pet = petService.getPet(petId);  
    model.addAttribute("pet", pet);  
  
    return "displayPet";  
}
```

```
@RequestMapping("/owners/**/pets/{petId}")
```



Herhangi derinlikteki path'leri kapsar

MVC Interceptor

- Servlet **Filter**'lara çok benzerler
- Ancak Filter'lar Servlet instance'ları öncesi veya sonrası devreye girer
- MVC Interceptor bean'leri ise Controller bean'lerinin **handler metotları** öncesi ve sonrası devreye girer
- **<mvc:interceptors>** elemanı ile bütün controller bean'larına bu xml elemanı içinde tanımlanan **interceptor bean'leri** **register edilebilir**

MVC Interceptor

```
public class SecurityInterceptor implements HandlerInterceptor {
```

```
@Override
```

```
public boolean preHandle(HttpServletRequest request,
    HttpServletResponse response, Object handler) throws Exception {
    String user = request.getRemoteUser();
    return user != null?true:false;
}
```

Handler metot invoke edilmeden
hemen önce çağrılır

```
@Override
```

```
public void postHandle(HttpServletRequest request,
    HttpServletResponse response, Object handler,
    ModelAndView modelAndView) throws Exception {
    //...
}
```

Handler metot invoke edildikten
hemen sonra, DispatcherServlet
daha view'ı render etmeden çağrılır

```
@Override
```

```
public void afterCompletion(HttpServletRequest request,
    HttpServletResponse response, Object handler, Exception ex)
    throws Exception {
    //...
}
```

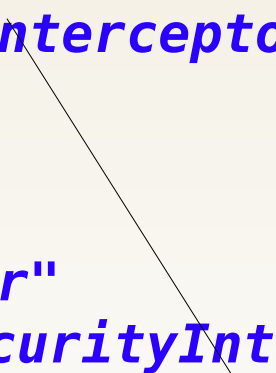
DispatcherServlet view'ı render
ettikten sonra çağrılır

```
}
```

MVC Interceptor Konfigürasyonu

```
<mvc:interceptors>
  <mvc:interceptor>
    <mapping path="/secure/*"/>
    <ref bean="securityInterceptor" />
  </mvc:interceptor>
</mvc:interceptors>

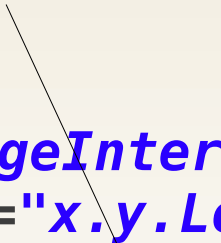
<bean id="securityInterceptor"
      class="x.y.SecurityInterceptor" />
```



Interceptor'ün sadece belirli request URI'larda devreye girmesi sağlanabilir

Global MVC Interceptor Konfigürasyonu

```
<mvc:interceptors>  
    <ref bean="localeChangeInterceptor" />  
</mvc:interceptors>  
  
<bean id="localeChangeInterceptor"  
      class="x.y.LocaleChangeInterceptor" />
```



Ya da global olarak bütün Controller bean'larının handler metotlarında devreye girmesi sağlanabilir

Java Tabanlı MVC Interceptor Konfigürasyonu

Spring 5 ile birlikte deprecated olmuştur
Yerine doğrudan WebMvcConfigurer arayüzü
kullanılabilir

```
@Configuration
@EnableWebMvc
public class WebAppConfig extends WebMvcConfigurerAdapter {

    @Autowired
    public SecurityInterceptor securityInterceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(securityInterceptor)
            .addPathPatterns("/secure/*");
    }
}
```

Request ve Session Scope'lar

- Spring Container **HTTP request ve session scope bean'leri** desteklemektedir
- **Request scope** bir bean instance'ının ömrü **web request'idir**
- Her web request'inde yeni bir bean yaratılır
- **Session scope** bir bean instance'ının ömrü ise **kullanıcı oturumu boyunca**dır
- Her farklı kullanıcı oturumu için yeni bir bean yaratılır

Request/Session Scope'lar ve Scoped Proxy

```
<bean id="userService" class="x.y.UserServiceImpl">  
    <property name="userPreferences"  
ref="userPreferences" />  
</bean>  
  
↓  
<bean id="userPreferences"  
class="x.y.UserPreferencesImpl" scope="session">  
    <aop:scoped-proxy />  
</bean>
```

Request veya session scope bir bean'in bağımlılık olarak enjekte edilebilmesi için `<aop:scoped-proxy/>` tanımı oldukça önemlidir!

Request/Session Scope'lar ve Scoped Proxy

```
<bean id="userPreferences"
class="x.y.UserPerferencesImpl" scope="session">
    <aop:scoped-proxy proxy-target-
class="false" />
</bean>
```



Default olarak class proxy oluşturulur, eğer **proxy-target-class="false"** ise interface tabanlı proxy yöntemi devreye girer
Bu durumda bean'ın en azından bir interface'i implement etmesi gerekir
Ayrıca bean bağımlılıklarının da interface üzerinden kurulmuş olması gerekir

```
<bean id="userService" class="x.y.UserServiceImpl">
    <property name="userPreferences"
ref="userPreferences" />
</bean>
```

Request/Session Scope'lar ve Scoped Proxy


```
@Component
@Scope(scopeName="session", proxyMode=ScopedProxyMode.INTERFACES)
public class UserPreferencesImpl implements UserPreferences {

    //...
}

@Service
public class UserServiceImpl implements UserService {
    private UserPreferences userPreferences;

    @Autowired
    public void setUserPreferences(UserPreferences userPreferences) {
        this.userPreferences = userPreferences;
    }

    //...
}
```



RequestContextListener Konfigürasyonu

- Spring Web MVC Framework (**DispatcherServlet**) kullanılırken scoped bean tanımlarının çalışması için **herhangi özel ayara gerek yoktur**
- Eğer istekler Spring Web MVC tarafından ele alınmıyor ise web.xml'de **RequestContextListener** tanımı yer almalıdır
 - JSF, Struts, Vaadin, GWT, Wicket vb.

RequestContextListener Konfigürasyonu

web.xml

```
<web-app...>
```

```
...
```

```
<listener>
```

```
<listener-class>
```

```
org.springframework.web.context.request.RequestContextListener
```

```
</listener-class>
```

```
</listener>
```

```
...
```

```
</web-app>
```



Vaadin, JSF, Struts, ZK gibi UI frameworkleri ile çalışırken scoped bean'ların çalışabilmesi için gereklidir

İletişim

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



harezmi
bilişim çözümleri

JAVA
Eğitimleri 