Spring Application Framework'e Giriş

harezmi bilişim çözümleri

Spring Nedir?

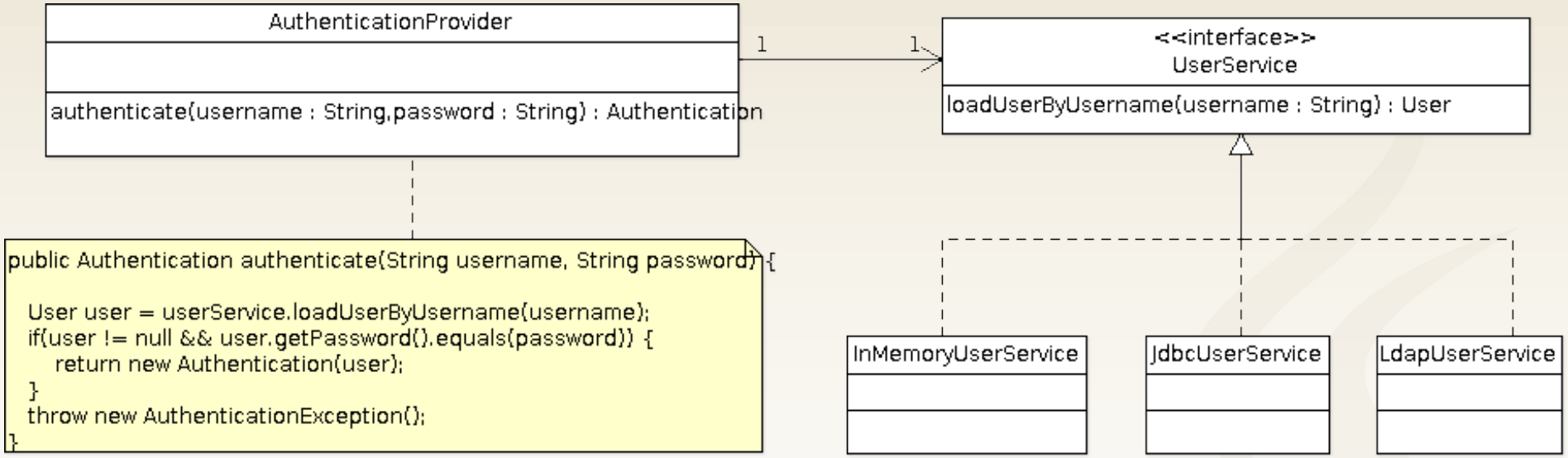
- **Kurumsal Java** uygulamalarını
  - **kolay**,
  - **hızlı** ,
  - **test edilebilir**

  biçimde geliştirmek ve Tomcat, Jetty, Resin gibi her tür ortamda çalıştırabilmek için ortaya çıkmış bir “**framework**”tür

Spring'i Öne Çıkaran Özellikler

- **POJO** tabanlı bir programlama modeli sunar
- **Program to interface** yaklaşımını temel ilke kabul etmiştir
- **Test edilebilirlik** her noktada **ön planda**dır
- **Modüler** **bir framework**tür, sadece ihtiyaç duyulan modüller istenilen kapsamda kullanılabilir

Program to Interface Yaklaşımı Nedir?



- Sınıflar arasındaki bağımlılıklar **arayüz ve soyut sınıfla**ra doğrudur
- Farklı ortamlardaki **farklı davranışlar** daha kolay biçimde ele alınabilir
- Bu sayede sınıfların **test edilebilir**liği de artmaktadır
- Testlerde **mock ve stub nesneler** daha rahat kullanılabilir

Spring IoC Container'a Doğru Yolculuk

JAVA Eğitimleri

Nesne bağımlılıklarını kendi içerisinde yaratır. Bu durumda farklı platformlar için farklı gerçekleştirimlerin yaratılması ve kullanılması problem olacaktır

```
public class AuthenticationProvider {
    private UserService userService =
                new JdbcUserService(new JdbcDataSource());
    private RoleService roleService = new RoleServiceImpl();

    public Authentication authenticate(
                String username, String password)
                                    throws AuthenticationException {
        User user = userService.loadUserByUsername(username);
        if(user != null && user.getPassword().equals(password)) {
            List<Role> roles =
                        roleService.findRolesByUsername(username);
            return new Authentication(user, roles);
        }
        throw new AuthenticationException("Authentication failed");
    }
}
```

Spring IoC Container'a Doğru Yolculuk

JAVA Eğitimleri

Nesne içerisinde basit bir factory metot içerisinde farklı platformlar için farklı gerçekleştirimleri oluşturma işlemi ele alınmaya çalışılır. Tabi bu bağımlılıkların da kendilerine ait bağımlılıkları vardır. Bunlarında ele alınması gerekecektir.

```
public class AuthenticationProvider {
    private UserService userService = createUserService();
    private RoleService roleService = new RoleServiceImpl();

    private UserService createUserService() {
        String targetPlatform =
                    System.getProperty("targetPlatform");
        if("dev".equals(targetPlatform)) {
            return new InMemoryUserService();
        } else if("test".equals(targetPlatform) ||
                    "prod".equals(targetPlatform)) {
            return new JdbcUserService(new JdbcDataSource());
        } else {
            return new LdapUserService(new LdapTemplate(
                                        new LdapContextSource()));
        }
    }
...
}
```

Spring IoC Container'a Doğru Yolculuk

Bağımlılıkları oluşturma ve platforma göre yönetme işi ayrı bir sınıfa çekilir. ServiceLocator isimli bu sınıf uygulama genelinde ihtiyaç duyulan bağımlılıklara erişim sağlar

```
public class ServiceLocator {

    private static final ServiceLocator
                                INSTANCE = new ServiceLocator();


    public static final ServiceLocator getInstance() {
        return INSTANCE;
    }

    private UserService userService;
    private RoleService roleService;
    private DataSource dataSource;

...
}
```

Spring IoC Container'a Doğru Yolculuk

ServiceLocator genellikle singleton olur ve bootstrap esnasında target platforma göre (dev,test veya prod) servis nesnelerini genellikle reflection api ile yaratarak kullanıma hazır hale getirir

```
public class ServiceLocator {
    private ServiceLocator() {
        try {
            String targetPlatform =
                    System.getProperty("targetPlatform");
            InputStream is = this.getClass().getClassLoader()
                .getResourceAsStream(
                    "service." + targetPlatform + ".properties");
            Properties properties = new Properties();
            properties.load(is);
            userService = (UserService) Class.forName(properties
                        .getProperty("userService")).newInstance();
            roleService = (RoleService) Class.forName(properties
                        .getProperty("roleService")).newInstance();
            dataSource = (DataSource) Class.forName(properties
                        .getProperty("dataSource")).newInstance();
            } catch (Exception ex) {
                throw new ServiceLocatorException();
            }
    }...
}
```

Spring IoC Container'a Doğru Yolculuk

```
public class ServiceLocator {
    ...

    public final UserService getUserService() {
        return userService;
    }

    public RoleService getRoleService() {
        return roleService;
    }

    public DataSource getDataSource() {
        return dataSource;
    }
}
```

Servisler getter metotlar üzerinden erişilebilirler. bazı ServiceLocator varyasyonlarında **getService(String serviceName)** gibi genel bir metot üzerinden de servislere erişim sağlanabilmektedir

Servis nesnelerini oluşturacak sınıfların FQN service.dev.properties gibi classpath'deki dosyalarda yönetilir

```
userService=com.javaegitimleri.example.JdbcUserService
roleService=com.javaegitimleri.example.RoleServiceImpl
dataSource=org.h2.jdbcx.JdbcDataSource
```

Spring IoC Container'a Doğru Yolculuk

```
public class AuthenticationProvider {

    private UserService userService =
                ServiceLocator.getInstance().getUserService();

    private RoleService roleService =
                ServiceLocator.getInstance().getRoleService();

    ...
}
```

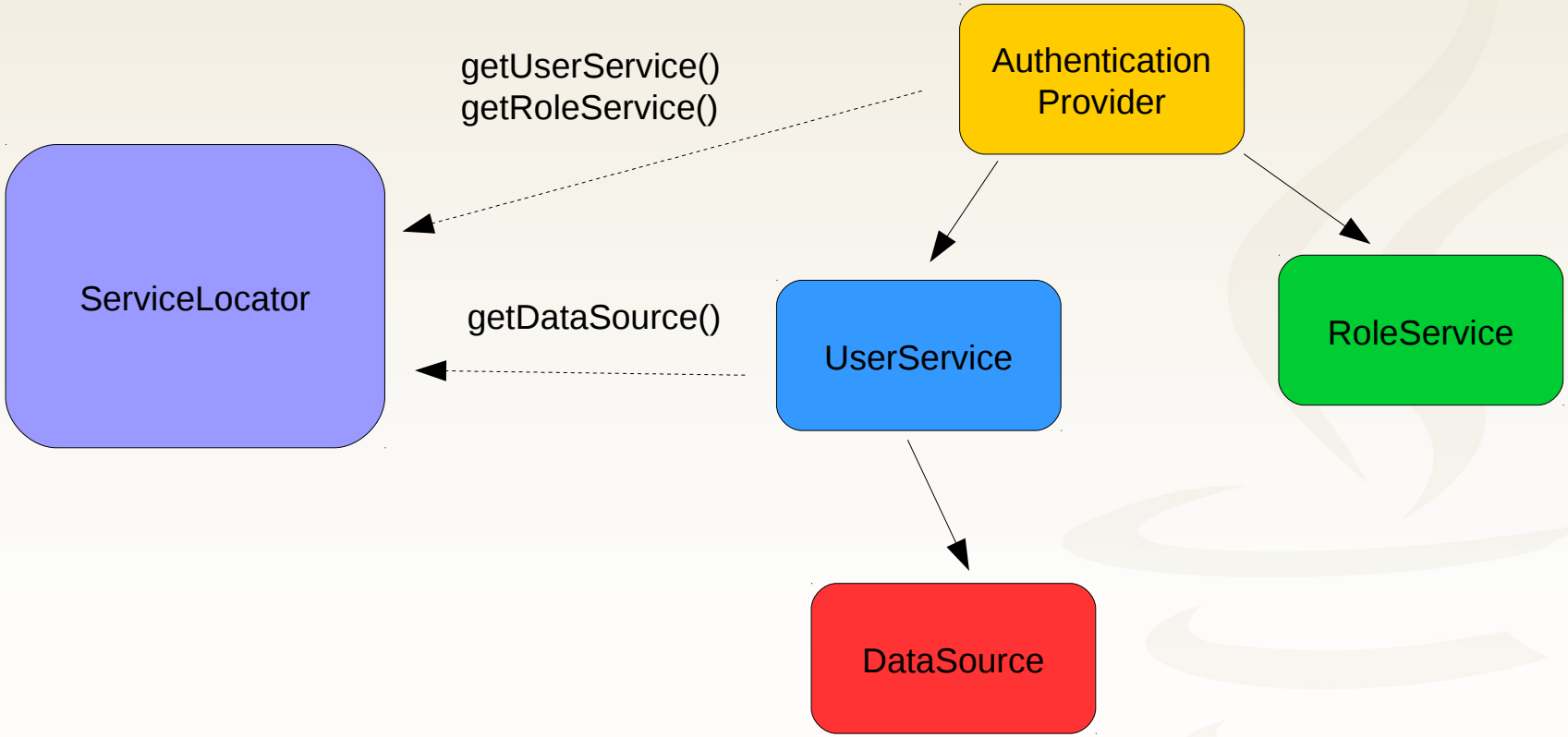
Nesneler ihtiyaç duydukları bağımlılıkları **ServiceLocator** üzerinden elde ederler

Spring IoC Container'a Doğru Yolculuk

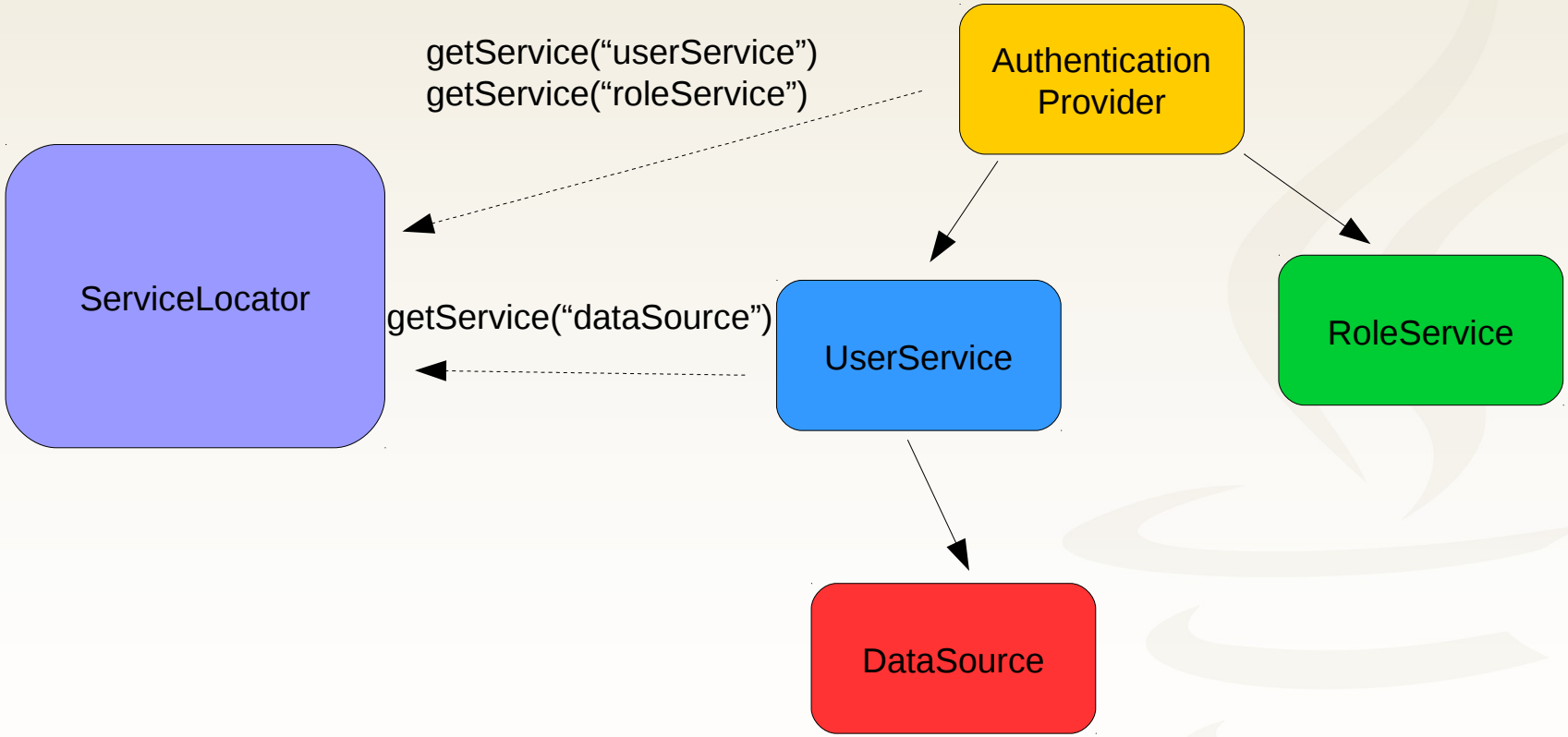
```
public class JdbcUserService implements UserService {

    private DataSource dataSource =
                    ServiceLocator.getInstance().getDataSource();

...
}
```

Bağımlılık hiyerarşisindeki bütün diğer nesneler de kendi bağımlılıklarını aynı şekilde ServiceLocator üzerinden çözümlerler

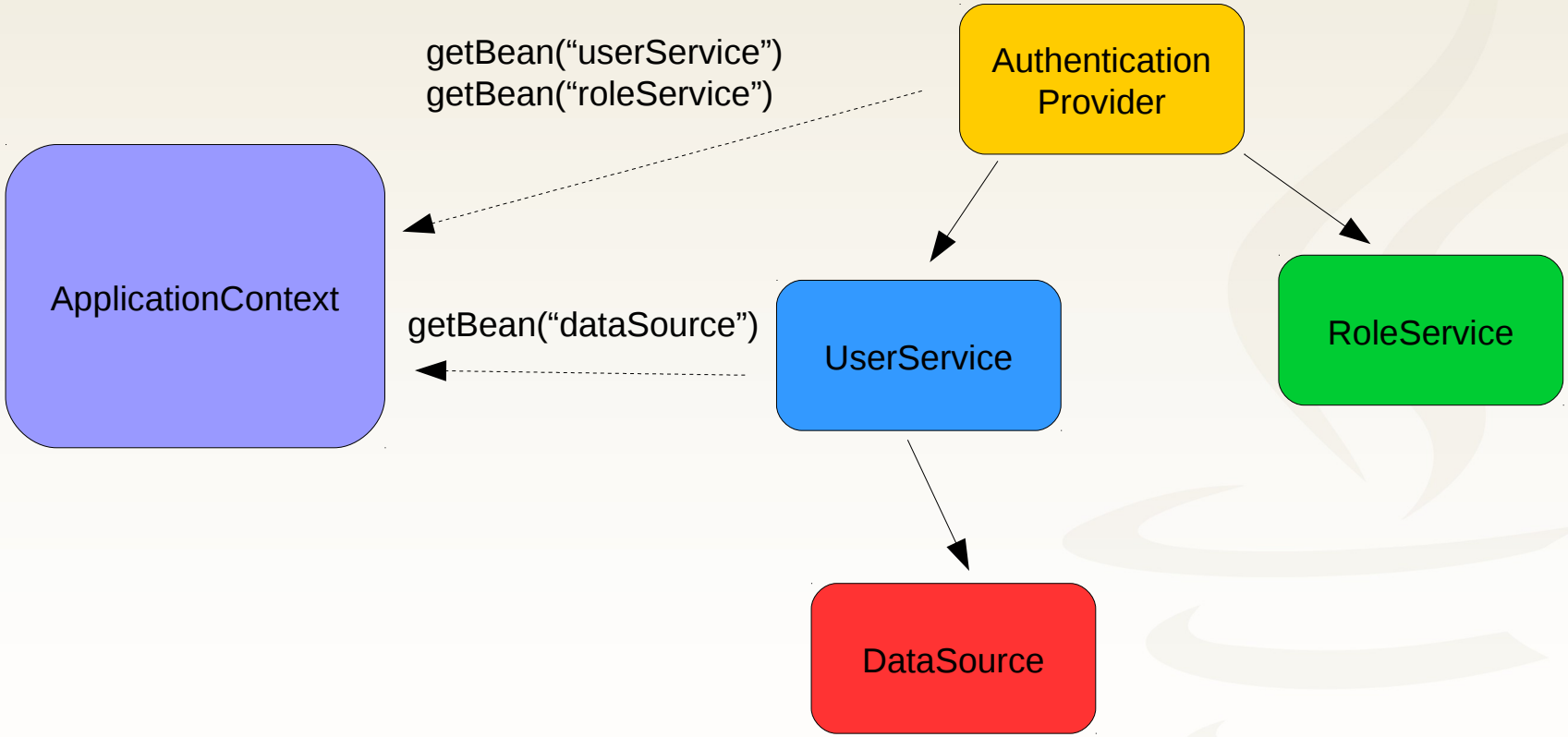
Spring IoC Container'a Doğru Yolculuk



Spring IoC Container'a Doğru Yolculuk



Spring IoC Container'a Doğru Yolculuk



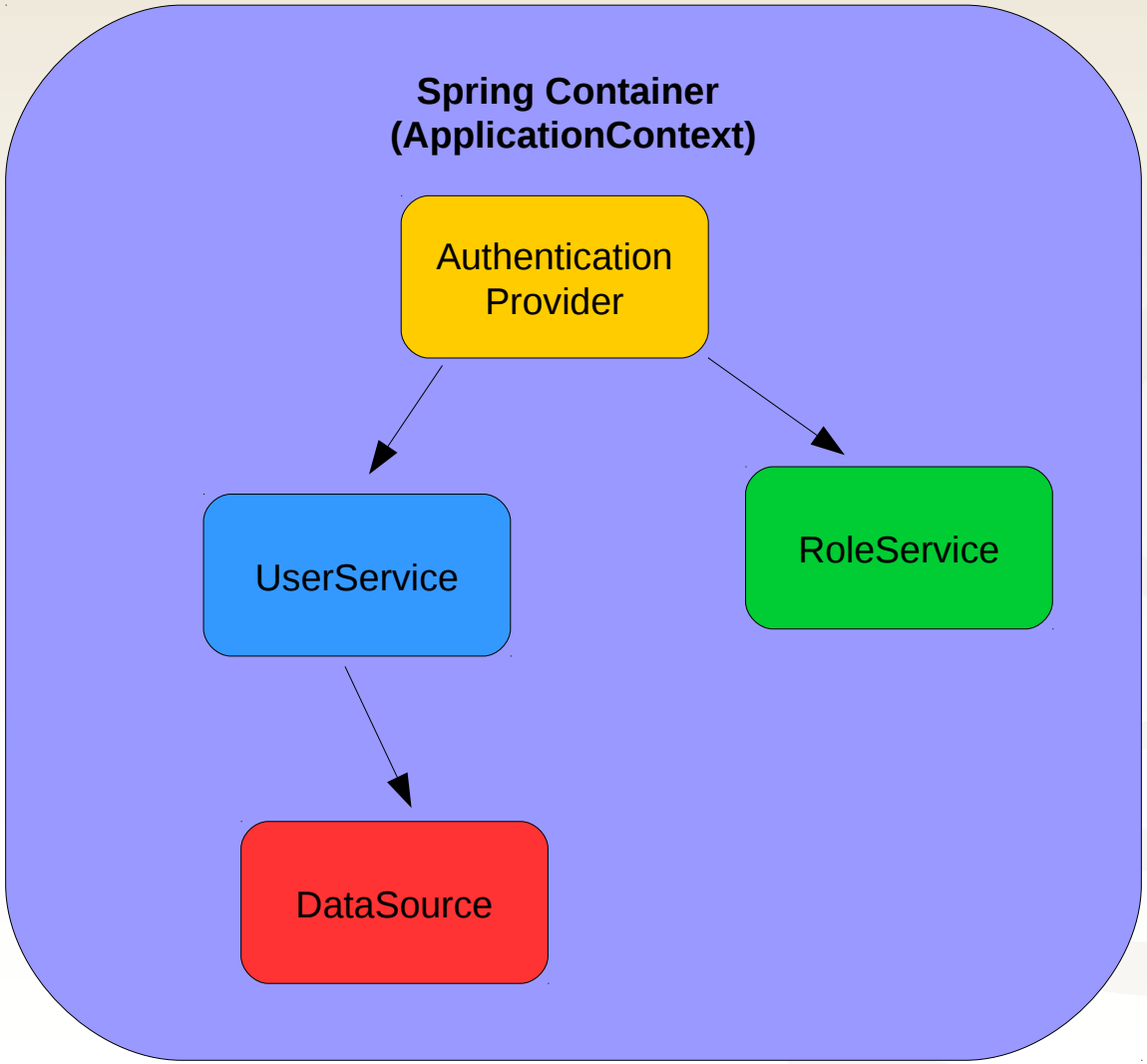
Spring IoC Container ve Dependency Injection

- Spring **nesneleri oluşturma, bir araya getirme ve yönetme işi**ne ServiceLocator'dan çok daha sistematik ve kapsamlı bir yol sunmaktadır
- Bağımlılıkları **oluşturma ve yönetme işi** nesnenin kendi içinden çıkıp, **Spring Container'a geçmektedir**
- Nesneler bağımlılıkların hangi **concrete sınıflar**la sağlandıklarını bilmezler

Spring IoC Container ve Dependency Injection

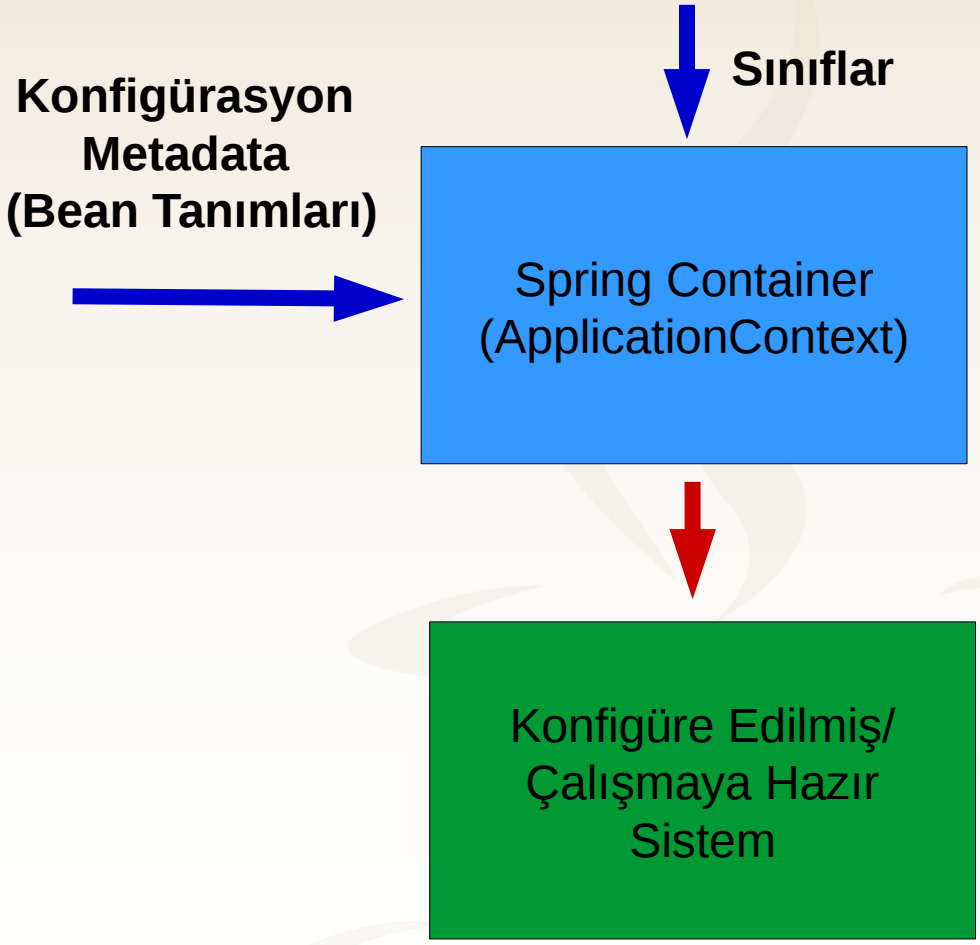
- Bağımlı olunan nesnelerin kim tarafından oluşturulduğu, nereden geldiği de bilinmez
- **Bağımlılıkların yönetimi nesnelerden container'a geçmiş**tir
- Spring IoC container tarafından yönetilen nesnelere “**bean**” adı verilir
- Spring bean'lerinin **sıradan Java nesneleri**nden hiç bir farkı yoktur

Spring IoC Container ve Dependency Injection



Spring IoC Container ve Konfigürasyon Metadata

- Bean'lerin yaratılması, bağımlılıkların enjekte edilmesi için Spring Container **bir bilgi**ye ihtiyaç duyar
- Bean ve bağımlılık tanımları **konfigürasyon metadata**'sını oluşturur



Spring IoC Container'ın Oluşturulması

- Spring Container'ın diğer adı **ApplicationContext**'dir
- ApplicationContext'i oluşturmak için **programatik** veya **dekleratif** yöntemler mevcuttur
- **Standalone** uygulamalarda **programatik** yöntem kullanılır
- **Web** uygulamalarında ise **dekleratif** yöntem kullanılır

Spring IoC Container'ın Oluşturulması

- XML, ApplicationContext oluşturmanın **geleneksel** yoludur
- Ancak **tek değil**dir
  - Java annotasyonları
  - Java kodu
- Spring IoC Container **konfigürasyon metadata formatından bağımsız**dır
- **Farklı metadata formatları**nı birlikte kullanarak da ApplicationContext oluşturulabilir

ApplicationContext'in Yaratılması ve Kullanımı

```
<beans...>
     <bean id="petClinicService" class="com.javaegitimleri.PetClinicService">
          <property name="ownerDao" ref="ownerDao"/>
     </bean>

     <bean id="ownerDao" class="com.javaegitimleri.OwnerDao"/>
</beans>
```

1

ApplicationContext konfigürasyon metadata kullanılarak yaratılır

```
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
```

2

Container bu aşamadan itibaren kullanıma hazırdır

Bean lookup ile ilgili bean'a erişilir

```
PetClinicService service = context.getBean("petClinicService", PetClinicService.class);
```

3

Artık bean'ler uygulama içerisinde istenildiği gibi kullanılabilir

```
List<Owner> owners = service.getOwners();
```

İletişim

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