Hibernate Konfigürasyonu

harezmi bilişim çözümleri

Hibernate ve Service Kavramı

- Hibernate 4 ve 5 sürümleri **Service** kavramı üzerine bina edilmiştir
- Hibernate'in persistence context, event yönetimi, JNDI, JMX, JTA gibi **bütün kabiliyetleri Service API'leri üzerinden** yönetilir ve istenirse özelleştirilebilir
- Bütün servisler **Service arayüzü**nden türer ve **kendi arayüzleri**ni tanımlarlar
- Herhangi bir servis arayüzünün farklı implementasyonları olabilir, bu sayede **pluggable bir mimari** ortaya çıkar

Service Lifecycle

- Bütün servislerin **standart bir yaşam döngüsü** vardır
  - Initiation
  - Configuration (opsiyonel)
  - Starting (opsiyonel)
  - In Use (ServiceRegistry açık olduğu müddetçe)
  - Stopping (opsiyonel)

Service ve ServiceRegistry İlişkisi

JAVA Eğitimleri

- Bütün bu servis nesneleri de **ServiceRegistry** üzerinden yönetilir ve erişilir
- ServiceRegistry, Hibernate servislerinin **yaşam döngüsü**nü işletir
- Servis nesneleri için bir nevi **IoC vazifesi** görür
- Servislerin ihtiyaç duyduğu **dependency'leri** ve **diğer servislere erişimi** sağlar

ServiceRegistry Türleri

- Hibernate konfigürasyonunda **temel olarak BootstrapServiceRegistry** ve **StandardServiceRegistry** nesneleri görev yapar
- Ayrıca SessionFactory içerisinde kullanılan bir **SessionFactoryServiceRegistry** de mevcuttur

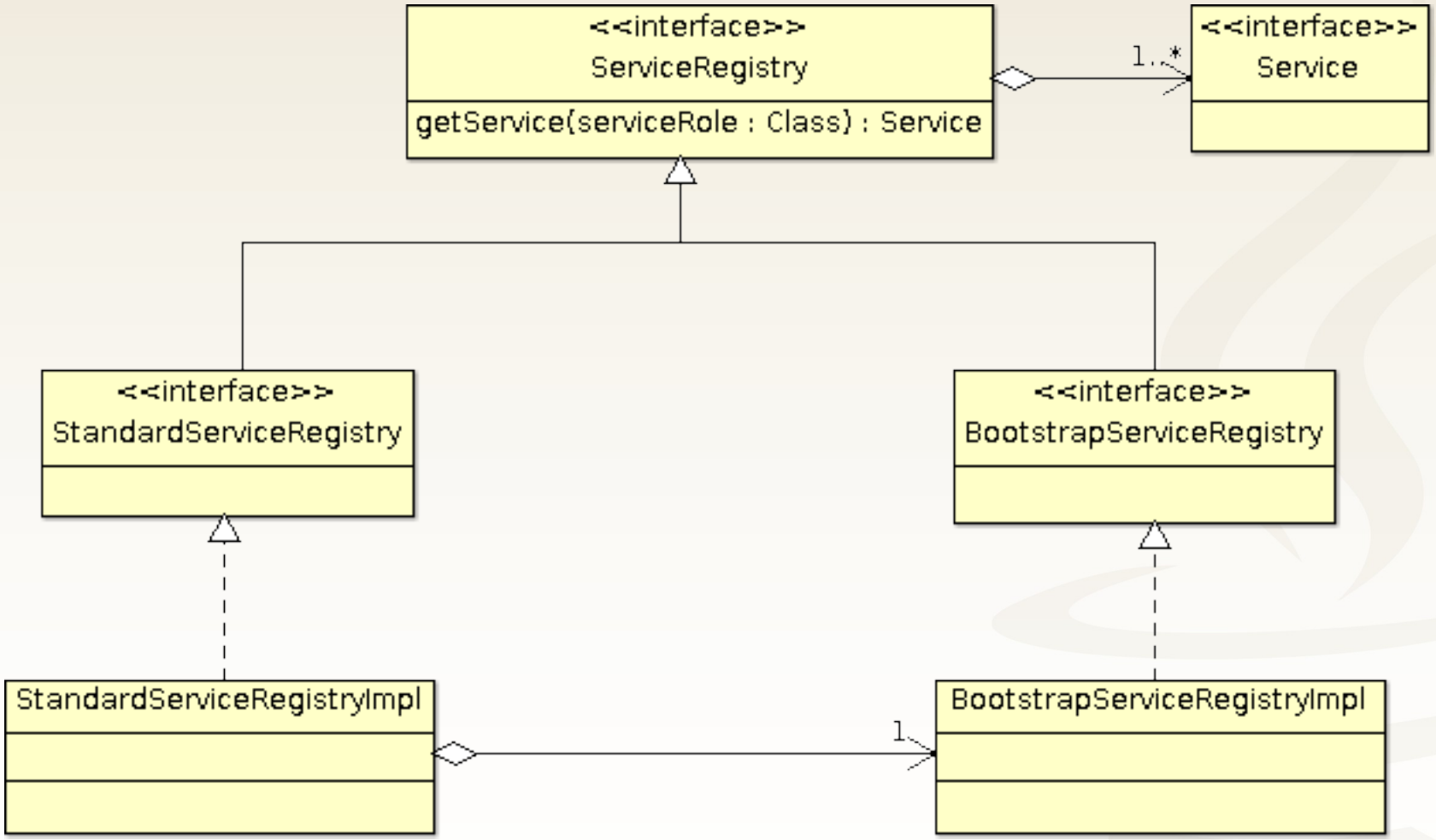
BootstrapServiceRegistry

- **BootstrapServiceRegistry** framework ile ilgili en temel servisleri sunar
  - **ClassLoaderService**: sınıf/servis yükleme, proxy nesne yaratma gibi işlemleri yapan servistir
  - **IntegratorService**: Hibernate ile entegre olacak modüllerin **Integrator** SPI nesnelerini sunan servistir
  - **StrategySelector**: Tanımlı **strategy** nesnelerinin register edildiği servistir

StandardServiceRegistry

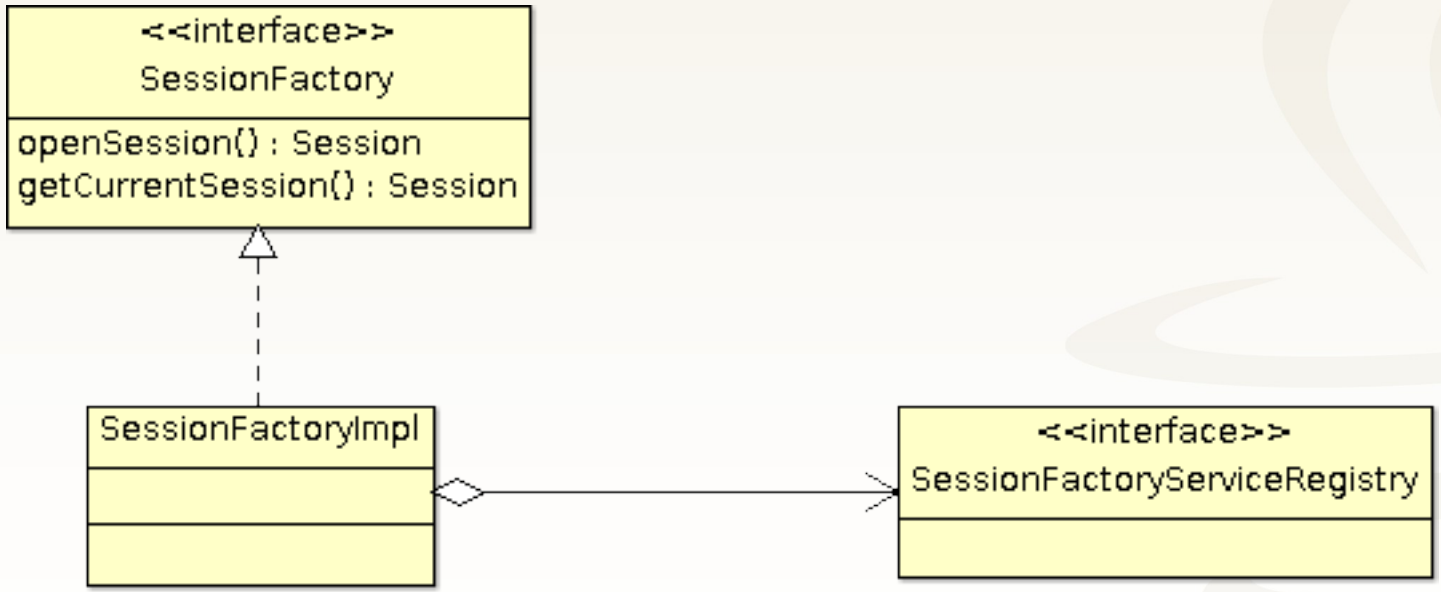
- Hibernate'in bütün diğer kabiliyetleri ise **StandardServiceRegistry** üzerinden sunulur.
  - **ConnectionProvider**: DB bağlantılarını yönetir
  - **DialectResolver**: Kullanılacak dialect nesnesini çözümler
  - **JdbcServices**: JDBC operasyonlarını tanımlar
  - **TransactionFactory**: Transaction yönetimini sağlar
  - **PersisterFactory**: Entity ve Collection Persister nesnelerini yaratır
  - ...

ServiceRegistry Hiyerarşisi



SessionFactory ve ServiceRegistry

- **SessionFactoryServiceRegistry** ise çalışma zamanında **SessionFactory** nesnesine erişim ihtiyacı olan servisleri yönetmek için kullanılır



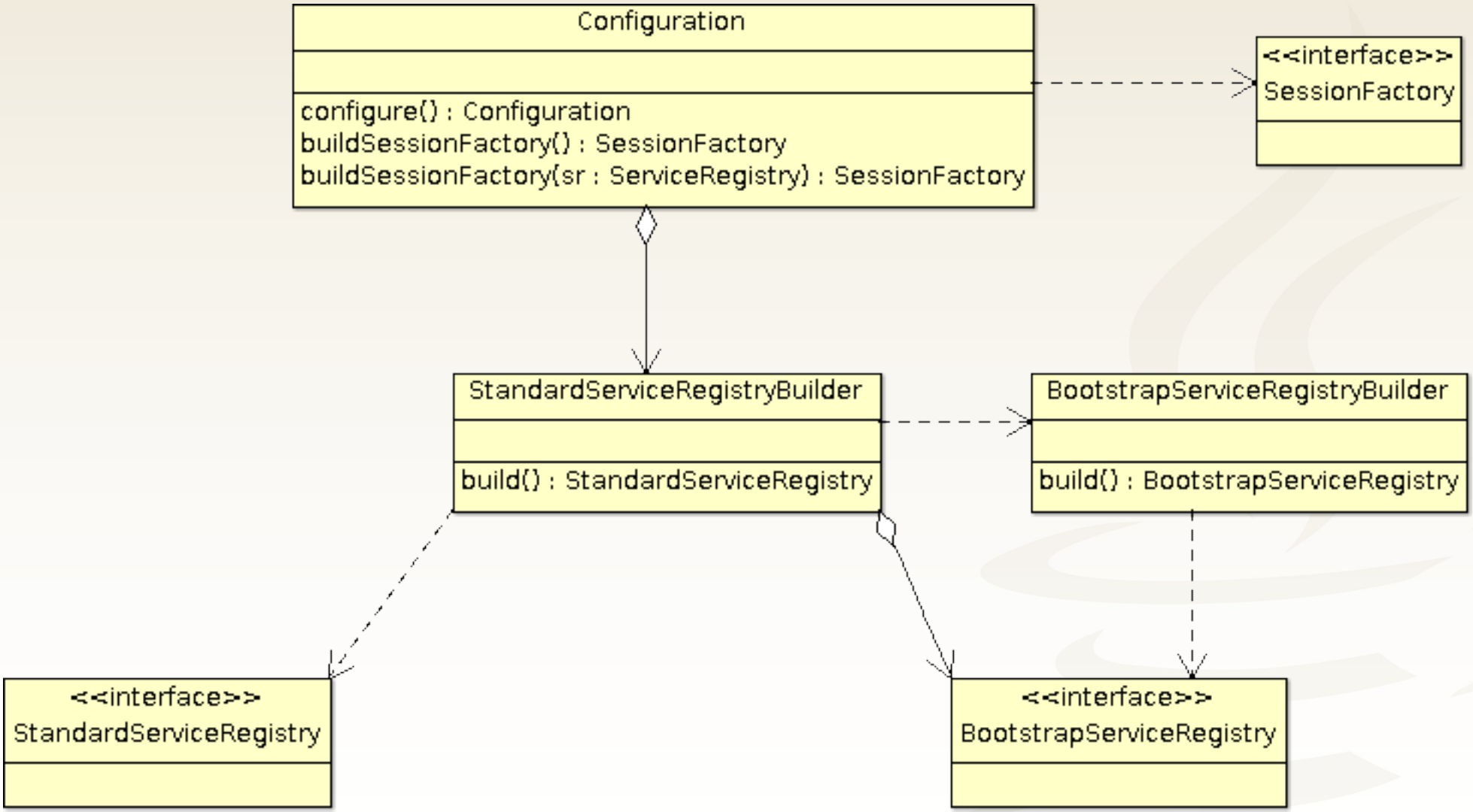
SessionFactoryServiceRegistry

- SessionFactoryServiceRegistry'nin yönettiği bazı servisler şunlardır:
  - **EventListenerRegistry**: Bütün Hibernate event listener'larını yöneten kısımdır
  - **StatisticsImplementor**: Hibernate'in Statistics API'sinin SPI kısmıdır

Hibernate Konfigürasyonu ve ServiceRegistry

- Hibernate'in runtime'da ayarlanması ve çalışır hale getirilmesi de tamamen bu **ServiceRegistry nesnelerinin oluşturulmasından ibaret**tir
- Configuration.**configure**() ve **buildSessionFactory**() metotları da kendi içinde **StandardServiceRegistryBuilder**'ı kullanmaktadır

Hibernate Konfigürasyonu ve ServiceRegistry



Hibernate Integrator SPI

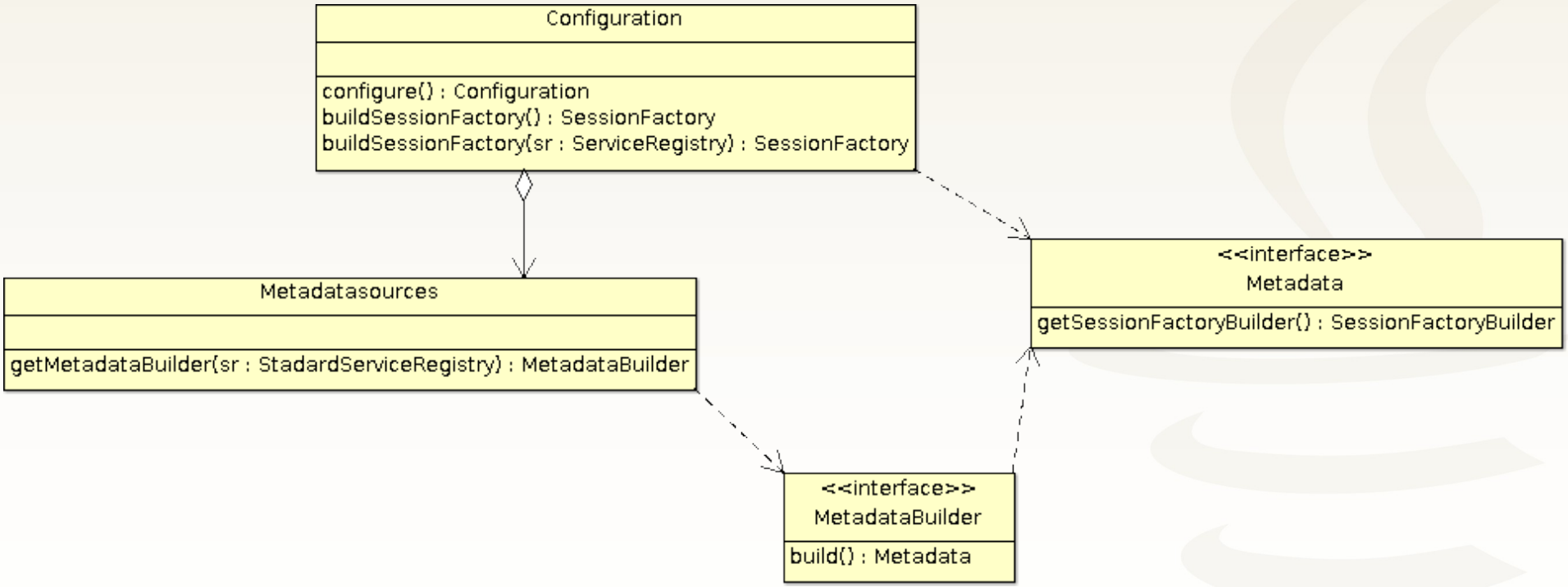
- Hibernate üzerinde özelleştirmeler ve ilave kabiliyetler eklemek için tanımlanmış **SPI arayüzü**dür
- Integrator instance'ları **SessionFactory initialization'ı** sırasında devreye girerler
- Hibernate Envers, Bean Validation vb kabiliyetler **Integrator vasıtası** ile devreye girerler

Hibernate Metadata

- Hibernate **Metadata** arayüzü ORM model'ini ifade eder
- Bütün mapping kaynakları (XML, annotations) tarafından yapılmış tanımların bir araya getirilmesi ile oluşturulur
- Configuration nesnesi tarafından **Metadatasources** ile XML, annotations gibi farklı metadata kaynakları process edilir
- Metadata, **MetadataBuilder** vasıtası ile oluşturulur

Hibernate Metadata ve SessionFactory

- Elde edilen Metadata üzerinden de **SessionFactory** nesnesi oluşturulur



Hibernate Konfigürasyon Bilgileri

- Configuration nesnesi, **konfigürasyon bilgilerinin nesne gösterimi** olarak da düşünülebilir
- SessionFactory oluşturulmadan önce Configuration'a **ilave mapping dosya bilgileri ve config tanımları** da eklenebilir
- Harici **konfigürasyon dosyaları olmaksızın** Hibernate SessionFactory konfigürasyonu yapmak da mümkündür

Konfigürasyon Dosyalarından Bağımsız SessionFactory Oluşturma

JAVA Eğitimleri

```
Properties settings = new Properties();

settings.put("hibernate.connection.driver_class", "org.h2.Driver");
settings.put("hibernate.connection.url", "jdbc:h2:tcp://localhost/~/test");
settings.put("hibernate.connection.username", "sa");
settings.put("hibernate.connection.password", "");
settings.put("hibernate.hbm2ddl.auto", "create");
settings.put("hibernate.show_sql", "true");
settings.put("hibernate.current_session_context_class", "thread");

Configuration cfg = new Configuration();

StandardServiceRegistryBuilder serviceRegistryBuilder =
                            new StandardServiceRegistryBuilder()
                                    .applySettings(settings).build()

SessionFactory sessionFactory = cfg.addProperties(settings)
    .addAnnotatedClass(PetType.class)
    .buildSessionFactory(serviceRegistryBuilder);
```

Konfigürasyon Dosyalarından Bağımsız SessionFactory Oluşturma

```
JdbcDataSource dataSource = new JdbcDataSource();
dataSource.setUrl("jdbc:h2:tcp://localhost/~/test");
dataSource.setUser("sa");
dataSource.setPassword("");

Properties settings = new Properties();

settings.put("hibernate.connection.datasource", dataSource);
settings.put("hibernate.dialect", "org.hibernate.dialect.H2Dialect");
settings.put("hibernate.show_sql", "true");
settings.put("hibernate.current_session_context_class", "thread");

Configuration cfg = new Configuration();

SessionFactory sessionFactory = cfg.addProperties(settings)
    .addAnnotatedClass(PetType.class)
    .buildSessionFactory(new StandardServiceRegistryBuilder()
            .applySettings(settings).build());
```

DataSource nesnesi Properties dosyasından da sağlanabilir. JNDI kullanmak şart değildir.

Hibernate Konfigürasyon Bilgilerinin Externalize Edilmesi

- Hibernate konfigürasyon property değerleri uygulama dışı bir lokasyondan elde edilebilir
- Property tanımlarının değerlerinde **placeholder** kullanılabilir
  - `<property name="show_sql">${displaysql}</property>`
  - -Ddisplaysql=true sistem property'si ile bu değer runtime da verilir
- Property tanımlarında **hibernate prefix**'i şart değildir

İletişim

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