Tasarım Örüntüleri
Factory Method

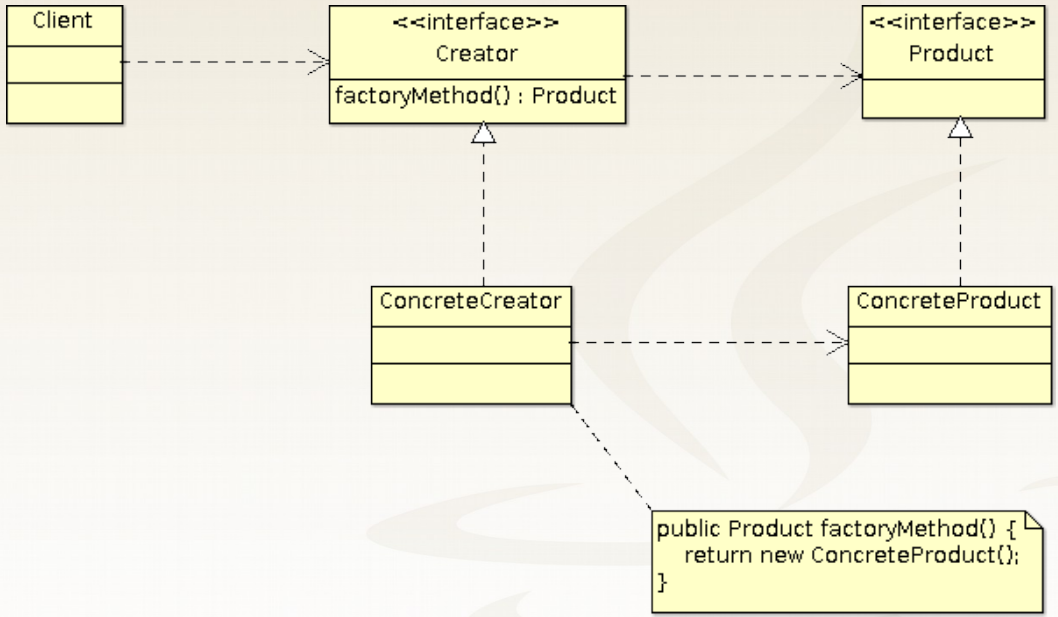
Örüntülerin Temel Prensipleri

- GoF tasarım örüntülerinin altında yatan **temel prensipler**
  - Encapsulation
  - Composition
  - Abstract Data Types

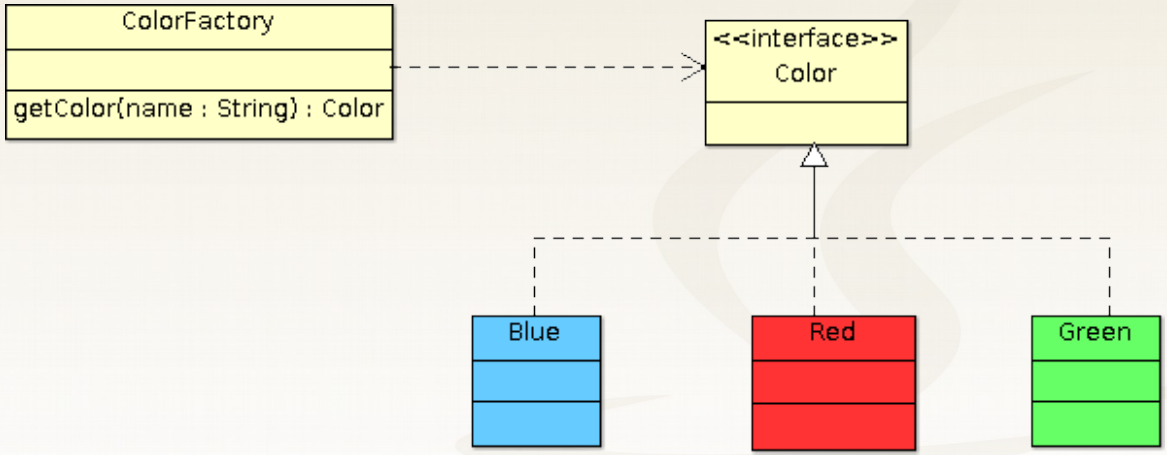
Factory Method Örüntüsü

- Factory Method
  - Nesne **yaratım süreci**ni encapsule eden bir örüntü
  - İstemci kodun ihtiyaç duyduğu nesneyi yaratan **ayrı bir sınıf**
  - Ve bu nesneyi yaratma işlemini gerçekleştiren **ayrı bir metot**
  - Yaratılan nesnenin **concrete tip**inin istemci tarafında bilinmemesi

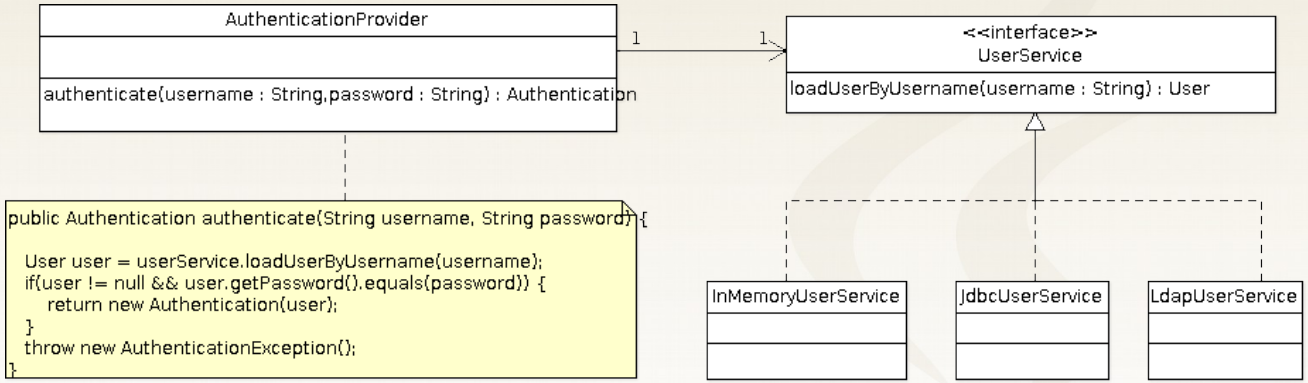
Factory Method Sınıf Diagramı



Parametrik Factory Method



Örnek Problem



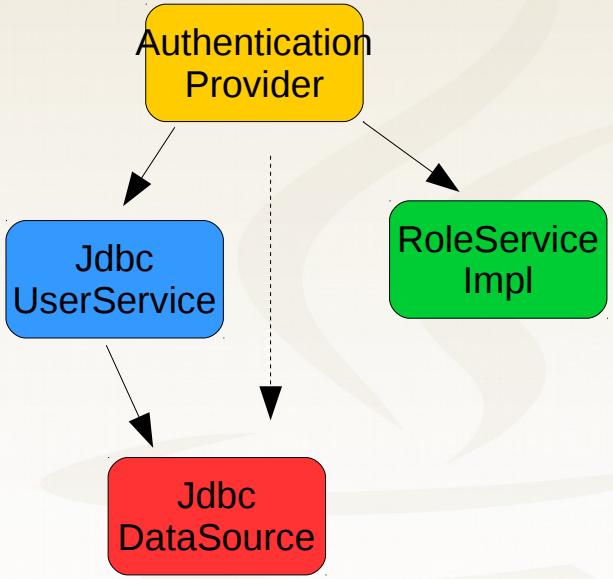
Örnek Problem

```
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");
    }
}
```

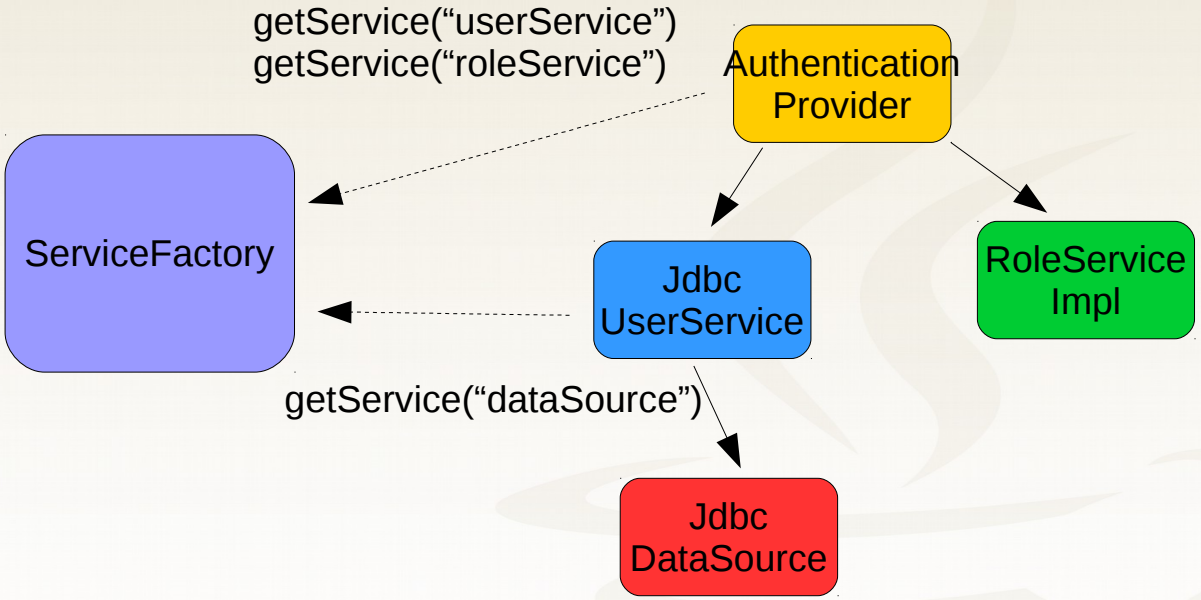
Örnek Problem

```
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()));
          }
     }
...
}
```

Örnek Problem



Parametrik Factory Method'a Geçiş



Servislere Erişim

```
public class ServiceFactory {

     ...

     private Map<String, Object> serviceMap = new HashMap<>();

     public Object getService(String name) {
          return serviceMap.get(name);
     }

     ...
}
```

Servislerin Yaratılması

```
public class ServiceFactory {
     ...
     private void initServiceMap() {
          try {
               String targetPlatform =
                         System.getProperty("targetPlatform");
               InputStream is = this.getClass().getClassLoader()
               .getResourceAsStream("service." + targetPlatform +
                                                   ".properties");

               Properties properties = new Properties();
               properties.load(is);

               properties.forEach((k,v)->{
                    String sName = k.toString();
                    String cName = properties.getProperty(sName);
                    Object service = Class.forName(cName)
                                                  .newInstance();
                    serviceMap.put(sName, service);
               });
          } catch (Exception ex) {
               throw new ServiceInitFailedException();
          }
     }
}
```

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

Servislerin Yaratılması ve ServiceFactory Nesnesine Erişim

```
public class ServiceFactory {

    private ServiceFactory() {
        initServiceMap();
    }

    public static final ServiceFactory INSTANCE =
                                        new  ServiceFactory();


    ...
}
```

Singleton ServiceFactory Nesnesine Erişim

```
public class AuthenticationProvider {

    private UserService userService =
        ServiceFactory.INSTANCE.getService("userService");

    private RoleService roleService =
        ServiceFactory.INSTANCE.getService("roleService");

    ...
}
```

```
public class JdbcUserService implements UserService {

    private DataSource dataSource =
        ServiceFactory.INSTANCE.getService("dataSource");

    ...
}
```

Spring ve Factory Method

- Spring **ApplicationContext**, nesnelerin bağımlılıklarını yaratmakta ve bunları ilgili nesnelere sunmaktadır
- Bu açıdan **kapsamlı bir factory method gerçekleştirimi**dir
- Spring **nesneleri oluşturma, bir araya getirme ve yönetme işi**ne ServiceFactory'den çok daha sistematik bir altyapı sunmaktadır

Spring ve Factory Method

- Spring tarafından yaratılan ve yönetilen nesnelere “**bean**” adı verilir
- Bağımlılık yönetimi nesnelerin kendisinden **Spring Container**’a geçmiştir
- Bu yönteme “**inversion of control**” adı verilmektedir
- Bu yöntemin diğer bir adı da “**dependency injection**”dır

Factory Method Örüntüsünün Faydaları

- Nesneyi **yaratan kısım ile nesneyi kullanan kısım** birbirinden tamamen ayrılmıştır
- Nesne **yaratma süreci** nesneyi kullanan yerden **bağımsız** biçimde yönetilebilir
- İstemci tarafının farklı concrete tiplerle **dinamik biçimde konfigürasyon**u mümkün hale gelir

İletişim

JAVA Eğitimleri

Kurumsal Java Eğitimleri


www.java-egitimleri.com


info@java-egitimleri.com


@javaegitimleri

youtube.com/c/
KurumsalJavaEğitimleri