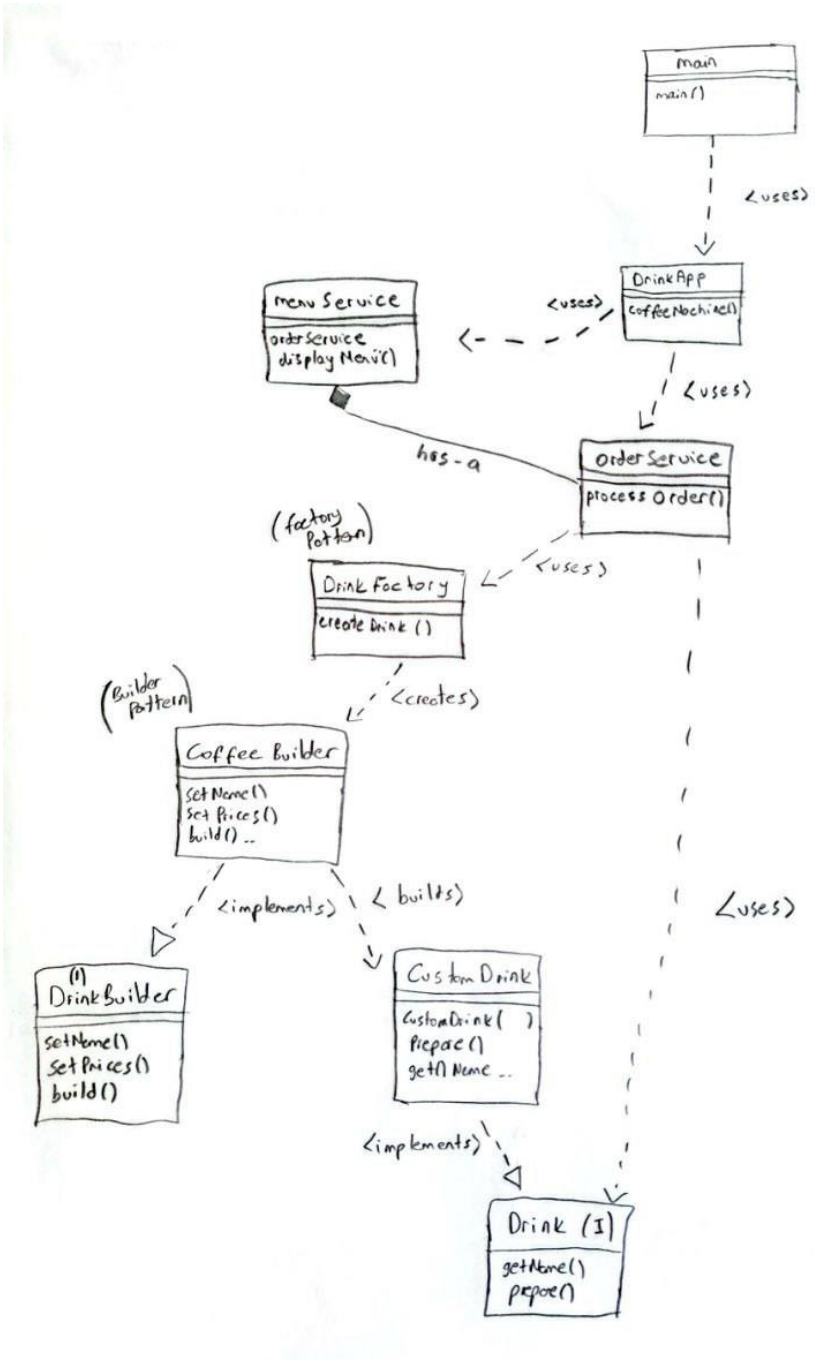
TASARIM



Tasarım sürecine başlamadan önce problemi iyice anlamaya çalıştım.Hangi tasarım kalıplarının proje isterlerine ve yapısına daha uygun olacağını düşündüm.Temel üretimm sürecinin yönetimi için **Factory design pattern** kullanmaya karar verdim. İçeceklerin yapısı da ele alındığında ilerde istenebilecek bir değişiklik ya da yeni bir formüle karşı cevap verebilmek için üretim sürecine dair geliştirmeler yapılmasına da karar verdim. Her alt parçanın fiyatı belli olsaydı bu çalışma **decorator design pattern** için iyi bir örnek olurdu ancak elimizde böyle bir veri yoktu.Bundan dolayı kahve çeşitlerinin üretiminde **builder design pattern** ı da kullanmaya karar verdim.

Bu yapıda Factory pattern üyenin seçimine göre o case e gidip ona göre builder design pattern kullanarak kahveyi build ediyor.

Single Responsibilty Principle (SRP) ye göre her sınıf yalnızca bir işten sorumlu olmalıdır. Bu yüzden menü yazdırma,kullanıcıdan alınan seçime göre DrinkFactory i çağırma ve yönetme gibi başlangıçta yapılması gereken işlemleri öylece main e koymadım.Main class **DrinkApp** class ını çağırarak işlemi başlatıyor.

DrinkApp classı **OrderService** ve **MenuService** nesnelerini oluşturduktan sonra **MenuService** için gerekli olan **OrderService** objesini daha az bağlı olmasını sağlamak için parametre olarak veriyor ve **MenuService** tarafında constructorda dependency injection tamamlanarak *menuService.displayMenu()* metoduyla menü işlemlerine başlanılıyor.

MenuService içerisinde kullanıcıdan input alındıktan sonra bu **OrderService** in *processOrder()* metoduna gönderiliyor.

processOrder() içerisinde **DrinkFactory** objesi oluşturuluyor. Ve **DrinkFactory** e ait *createDrink(selection)* metodu çağırılarak dönüş tipi Drink tipi bir değişkene atanıyor.Sonrasında gelen objenin *.prepare()* ve *.getPrice() metodları ile bilgiler ekrana bastırılıyor.*

Şimdi gelelim **DrinkFactory** de nesnenin üretim sürecine, içeride **DrinkBuilder** türünde bir **CoffeeBuilder** objesi oluşturuluyor.DrinkBuilder burada temel içecek imzalarını içeren interface ve **CoffeeBuilder** bu interface i implemente ediyor.

Sonrasında switch-case yapısı ile kullanıcının tercihine göre hangi kahveyse onun alanına gidilerek oluşturulan **CoffeeBuilder** ın builder metodları çağırılıyor ve o türü özgü içerikleri ekleniyor.

CoffeeBuilder ın metodları içindeki *name, prices ve içerik listesine* ekleme yapan sınıflardan oluşuyor ancak her bir eklemeden sonra *returrn this* diyerek mevcut objelerini return ediyor ve çağırılan diğer metodla devam edilmesini sağlıyorlar.Son olarak *build()* metodu çağırıldığında **Drink** sınıfını implemente eden **CustomDrink** sınıfını constructordan bilgileri vererek oluşturup return ediyor.

Böylece return edilen bu bu obje önce **DrinkFactory** e oradan da **processOrder** a kadar ulaşıyor ve bilgileri yazdırılıyor.

TEST

Test süreci sorunsuz yönetebilmek ve birim,entegrasyon testleri yazmak için projeyi **maven** projesi olarak düzenlemiştim.Bu yüzden *pom.xml e* **junit** bağımlılığını ekledikten sonra test sınıflarımı oluşturmaya başladım. ***CoffeeBuilderTest*** sınıfında *testCreateEspresso*() ile DrinkBuilder ın çalışmasını kontrol ettim. *testSpecialCharactersInName()* ile farklı karekterlerde sorun oluyor mu diye kontrol ettim ve *testMaximumPrice()* maximum integer değerine kadar bir sorun olmadığından emin oldum.

DrinkFactoryTest sınıfında factory sürecine dair *testCreateCoffee_ValidInput* ile inputları kontrol ettim.Yine *testCreateDrink_NegativeInput()* ile negatif inputları,*testAllDrinkTypes()* ile tüm typların kontrolünü sağladım.

SystemIntegrationTest ile birimler arası entegrasyon testleri hazırladım. *testCompleteOrderFlow() testi tüm akışı kontrol ediyor.Bunu sağlamak için input ve output simulasyonunu sağlayan objeler hazırlandı.Verilen inputtan sonra assertTrue(output.contains(…)) ile outputun içeriği kontrol edilerek test tamamlanıyor.*

Yine **SystemIntegrationTest** altında *testMenuOrderServiceIntegration altında Menu ve Order testlerinin ilişkisi kontrol ediliyor.*

Javadoc Ve Yorumlar

Proje boyunca tüm class lara ve ve metodlara javadoc dökümantasyonlamasına uygun olacak şekilde yorumlar eklendi. Değişkenlere ve metodların dönüş tiplerine ek açıklamalar getirildi.Üretilen javadoc dökümanı da proje içerisine eklendi.

Git ve versiyonlama

Proje boyunca farklı branchlere kodlar belirli aralıklara git bash kullanılarak pushlandı.Bu branchlerden de işlevini tamamlayanlar merge edilerek main branche aktarıldı.

Github Url: https://github.com/DogukanTastan/bilgem-work

Doğukan TAŞTAN

dogkantastan29@gmail.com