CSE 443 - Object Oriented Analysis and Design
Homework2 Rapor

Furkan YILDIZ 141044031

Part1)

1) Eğer singleton sınıfı ya da onun parent'i "Clonable" interfacesini implement etmemişse, singleton objesinin "clone" methodu çağırıldığında Java compiler error vericektir çünkü "clone" methodu protected tır.

2) "java.lang.Cloneable" interfacesi implement edilerek, Singleton class'ı için clone methodu override edilir. Bu methodun içerisinde ise "CloneNotSupportedException" fırlatılır.

2) Cevap Parent Class'ın Clonable interfacesinin methodlarını implement edilişine göre değişir,

a)

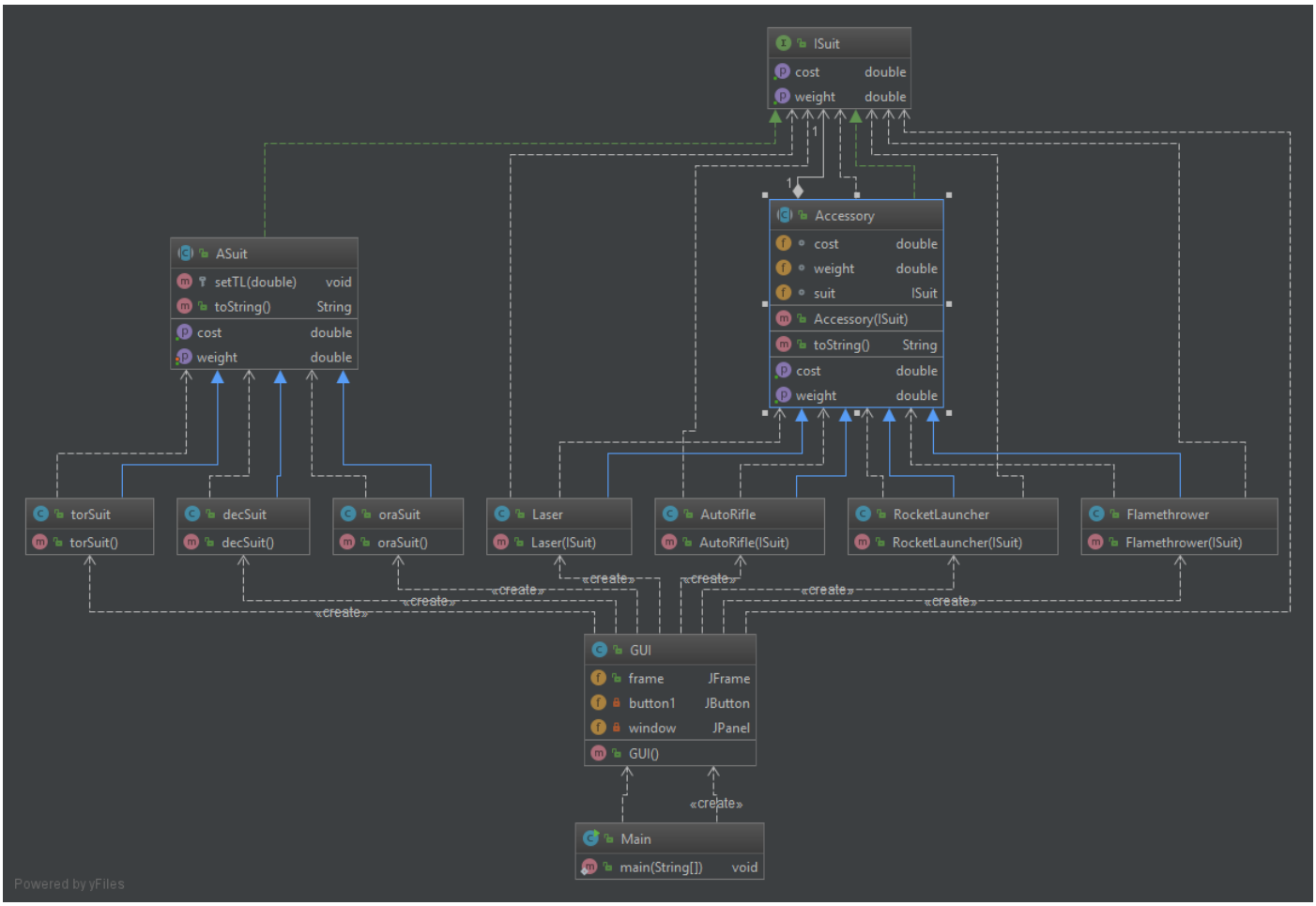
Eğer Parent class'ının içerisinde clone method'unda object' in clone methodu çağırılırsa, ya da instance kopyası yapılarak, yapılan bu kopya return edilirse, singleton objectten ikinci bir obje clone edilebilir hale gelecektir. Yani Singleton'luk özelliği bozulacaktır.

Eğer Parent class'ının içerisinde clone method'unda "CloneNotSupportedException" fırlatılırsa, ikinci bir obje oluşturulamayacaktır.

b)

Önlemek için Parent Class' ta ya da Singleton class' ımızda "clone" methodunu override ederek "CloneNotSupportedException" fırlatırız.

Part2)



Suit Interfacesinin getCost ve getWeight methodları var. Bu interface ASuit sınıfı tarafından implement edilerek, fonksiyonların kodunu oluşturuyor. Suit sınıfları (tor,dec,ora) ise contructurelerinde bu sadece TL ve

weight'i set ederek hayatlarına devam ediyorlar. Suit sınıflarına kendi structurelerini değiştirmeden yeni özellikler katmak istiyoruz. Bu istek bizi "Decorator Design Pattern' a" götürüyor.

Suit türlerine yeni özellikler ekleyebilmek için, Suit interfacesini implement eden Accessory abstract sınıfı oluşturuyoruz. Bu sınıfta özellik eklemek istediğimiz Suit' in referansını tutuyoruz. Suit Interfacesindeki getCost ve getWeight methodlarını Suit'in getCost ve getWeight larına accessory'in cost ve weigt'ini ekleyerek return ediyoruz. Böylece saf Suit' in cost 'u + yeni özelliğin cost' u yapmış oluyoruz. Suit' e eklemek istediğimiz aksesuarları ise Accessory sınıfından türeterek constructurelerinde cost, weight i implement ediyor, Suit'in referansını kaydediyoruz.

Böylece örnek verirsek,

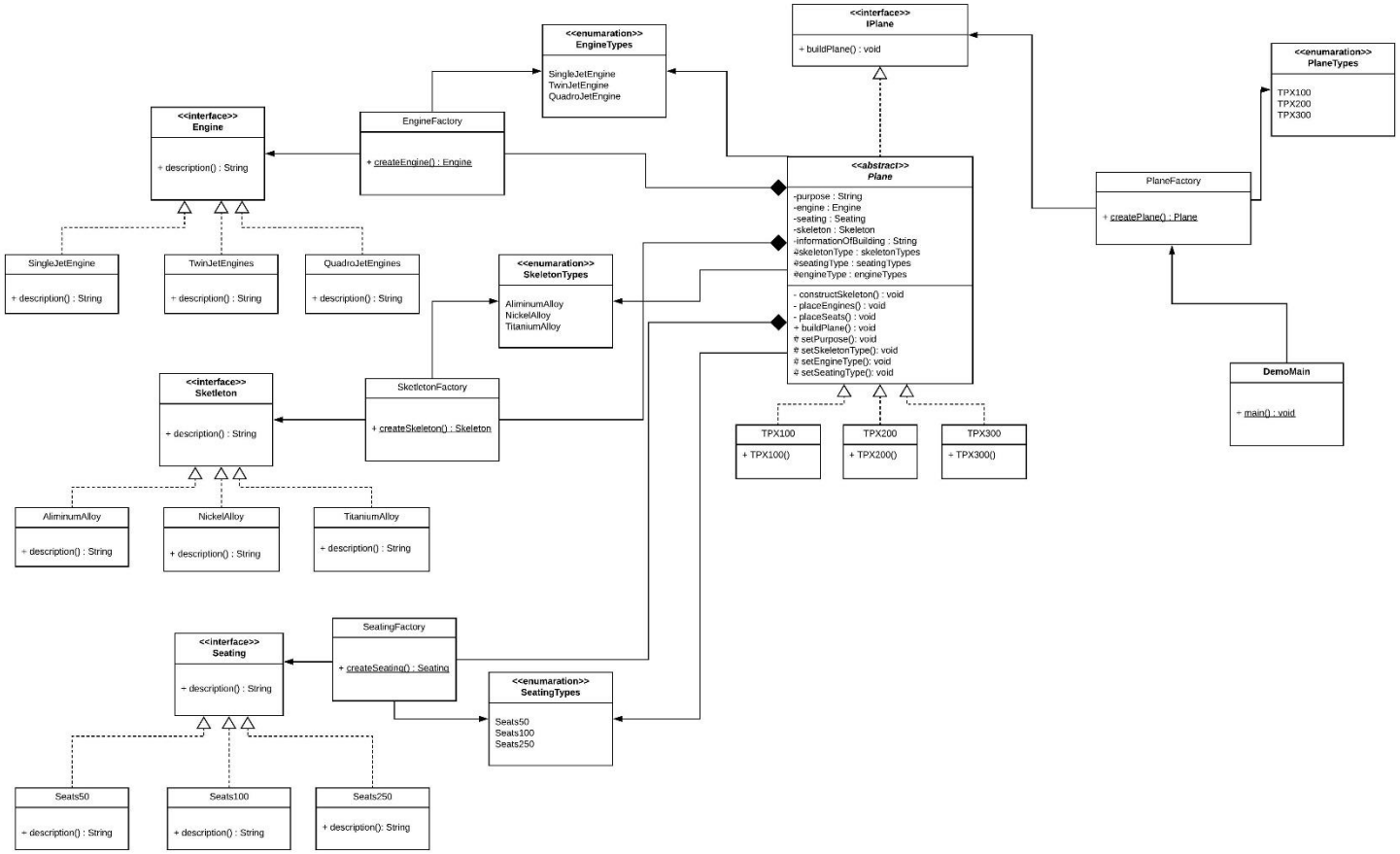
"a dec suit with 1 flamethrower, 2 automatic rifles and 1 rocket launcher" oluşturmak için

new RocketLauncher(new AutoRifle (new AutoRifle(new Flamethrower(new decSuit()))))

dememiz yeterli oluyor.

Bunu GUI üzerinden yapmak için ISuit referansı tutularak önce GUI'den istenilen suit, daha sonra istenildiği kadar aksesuar ekleniyor. Aksesuarlar için maksimum sınır 20' şer olarak tutuldu. Bu sınır computational calculation error' e düşmemek için yapıldı.

Part3)



Factory Design Pattern ‘i kullanıldı.

Engine,Skeleton ve Seating için interface oluşturularak bu interfacelerden bu sınıfların tipleri oluşturuldu. (Engine interfacesinden, Engine tipi olan SingleJetEngine,TwinJetEngines,QuadroJetEngines gibi) Aslında basitçe main bir class oluşturularak bunlar class variablesi olarakta tutulabilinirdi, sorulan soru için bu çözüm yeterlide olurdu fakat TAI’ de çalıştığımızı düşünürsek, bu tipler string gibi bir class variablesi olarak tutulamaz çünkü içerilerinde birçok method,variable bulundurmaları gerekir.

Engine,Skeleton ve Seating sınıflarının içerisine “description” methodu eklendi. Her sınıf decription’da ne yaptığı hakkında console ‘ye bir cümlelik bilgi yazıyor. (Örneğin TPX100 modeli oluşturulduğunda Skeleton olarak Aluminum alloy kullanılıyor. Aluminum Alloy kullanıldığında ise console ye “Aluminum alloy created.” Şeklinde bilgilendirici bir mesaj basılıyor.) ayrıca bunu GUI’de gösterebilmek içinde bu stringi return ediyor. Bu şekilde programın akışı takip edilebiliniyor.

Engine,Skeleton, Seating ve Plane sınıflarının tipleri için enumarator oluşturuldu. Böylece hatalı giriş yapılması önlendi. (String olsaydı yanlış yazım olabilirdi vs.)

Abstract bir Plane sınıfı oluşturuldu. “buildPlane” methodunda uçağın yapım aşamaları olan constructSkeleton, placeEngines, placeEngines methodları çağırıldı. Bu methodlar ise ilgili fabrika sınıflarını çağırarak run time da tiplerine göre Engine,Skeleton ve Seating nesnelerini oluşturuyorlar. (Örneğin constructSkeleton

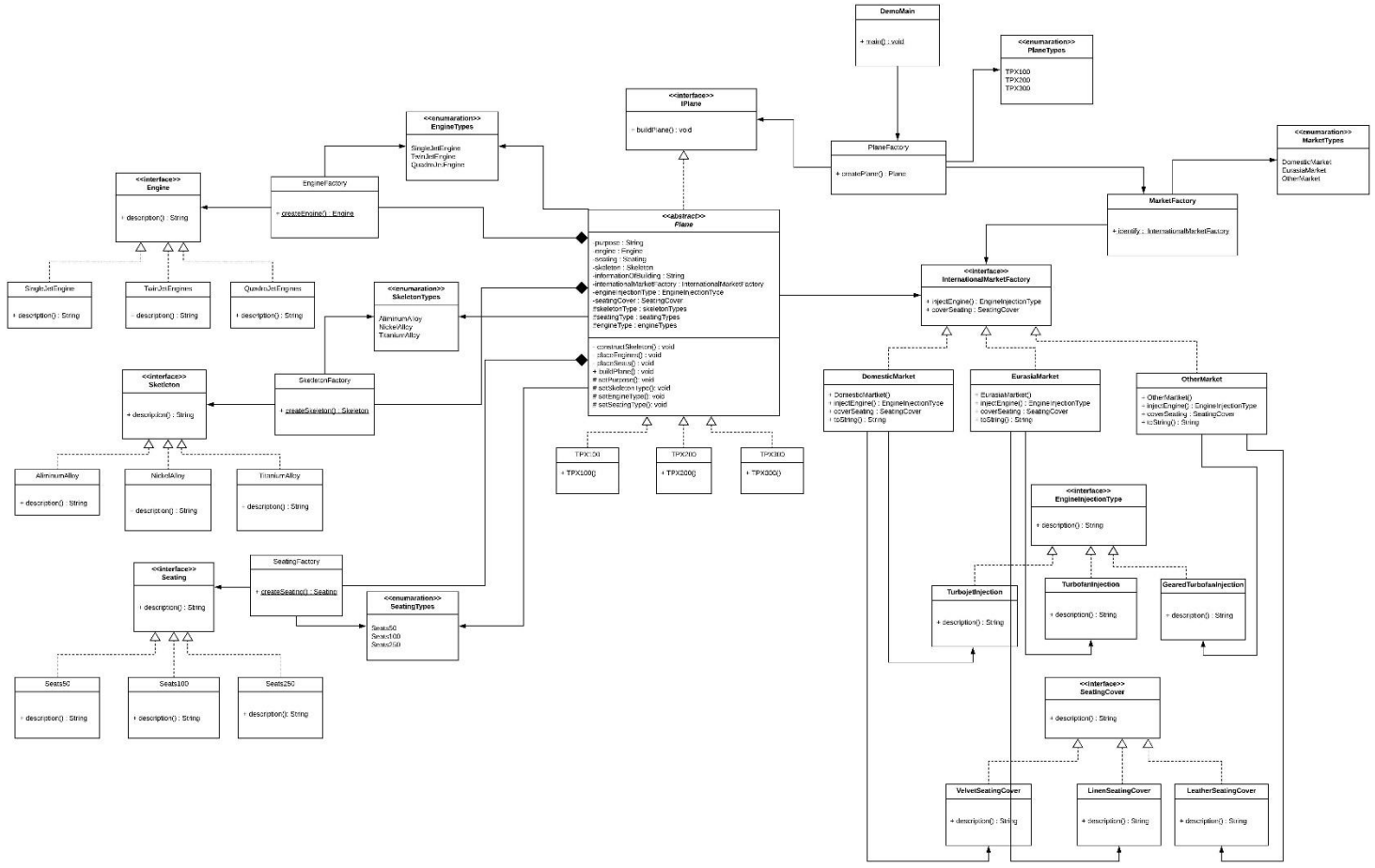
sınıfı Skelethon fabrikasını çağırıyor ve Uçağın modeline göre ilgili Skelethon' un nesnesi oluşturularak return ediliyor.

Plane sınıfından türetilen uçak modelleri ise constructor larında modellerine göre koyulması gereken Engine,Skeleton ve Seating tiplerini tanımlanıyor.

PlaneFactory sınıfı yazılarak içerisinde static "createPlane" methodu eklendi. Bu method model tipi alıyor ve bu tipe göre model oluşturuyor.

UML diagramının karmaşık görünmeyip, anlaşılabilir kalması için GUI, eklenmedi.
Program çalıştırıldığında GUI'den uçağın tipini seçtikten ve "Simulate!" butonuna basıldıktan sonra yapılan işlemlerin sırası ile yazdığı bir bilgi kutucuğu geliyor ekrana.

Part3 b)



Market tipine göre ürün çeşitliliği istendiğinden factory methodu bize yetersiz hale geliyor ve bu sebeple abstract factory methoduna başvuruyoruz. Bu design pattern'ine göre Engine Injection için bir interface oluştuurluyor ve tipleri olan EngineInjection, TurbofanInjection, Geared Turbo Fan Injection sınıfları bu interfaceden implement ediliyor. Aynı şekilde Seating Cover içinde interface oluşturularak tipleri bu interfaceden implement edilerek class oluşturuluyor.

InternationalMarketFactory adında bir interface oluşturularak bu interfaceye "injectEngine", "coverSeating" methodları ekleniyor. Bu methodlar, bu interfaceden implement edilecek class lar tarafından oluşturulacak.

Domestic, Eurasia ve Other sınıfları InternationalMarketFactory interfacesinden implement edilerek oluşturuluyor. Bu sınıfların görevi kendi bölgeleri için gerekli tipte motor ve koltuk üretmek. Bu sınıfların içerisindeki "injectEngine" methodu ilgili tür için EngineInjectType üretmekle sorumlu (Örneğin Domestic'de TurbojetInjection üretirken, Eurasia da TurbofanInjection, Other de GearedTurbofanInjection üretecek.)
"coverSeating" methodu ise ilgili tür için SeatingCover üretmekten sorumlu.

Plane içerisinde InternationalMarketFactory tipinde bir referans tutarak, ilgili plane'nin, ilgili marketinin gerektirdiği ürünleri üretmeyi garanti ediyoruz. Plane objesi oluşturulurken InternationalMarketFactory'in referansı veriliyor.

MarketFactory sınıfı yazılarak içerisine static "identify" methodu eklendi. Bu method marketin tipini alarak, gerçerli market tipinin objesini return ediyor. Static yapılmasının nedeni, bu sınıfın objesini tutmaya gerek olmaması.

PlaneFactory sınıfı yazılarak içerisinde static "createPlane" methodu eklendi. Bu method model tipi ve market tipi alıyor. MarketFactory sınıfının static identify methodunu çağırarak market objesini oluşturuyor. Bu objeyi ise oluşturulacak model' e veriyor (consructure ile). Böylece model o markete göre oluşturuluyor.