**Singleton vs. Prototype Beans**

• If a bean is defined as a singleton then it can have at most one instance and that instance is shared with all other beans into which it is injected.

• All beans that receive the singleton collaborator bean via injection have the same instance, which becomes the shared state among the beans.

• If a bean is defined as a prototype then every bean into which it is injected has its own instance of the collaborator bean.

• In this case the instances of the collaborator bean is not shared, every instance of the collaborator bean is part of the private state of a bean it is injected.

**Singleton olduğu zaman inject edilen her yerde ortak olarak kullanılıyor demektir.**

**Prototype olursa inject edilen her birine ayrı ayrı inject ediliyor.**

• So when to use singleton and when to use prototype?

• Of course in terms of run-time efficiency less objects would be much better so the singleton would be the choice.

• But most of the time the run-time efficiency is not the unique factor that determines the choice.

• The key term on this issue is the state, i.e. whether the collaborator bean has a state and if it does then whether that state can be changed by the beans into which it is injected and the environment is multi-threaded.

**Singleton ise share state’dir hepsinin ortak parçasıdır.**

**Örnek olarak e-ticaret sitesine kullanıcı her giris yaptığında yeni bir obje olusturmasını istediğimiz zaten prototype olmus olur ama her giriste bi obje olusuyorsa singleton, tabi bu web örneğini baz alırsak Scope türlerinden session da kullanılır. Anlasılması icin böyle basic örnek verildi.**

• If the collaborator bean does not have a state, i.e. it is stateless not statefull, it shoud be a singleton.

• All method calls made to the collaborator bean can only use the local variables which are thread-safe.

• If the collaborator bean does have a state, i.e. it is statefull not stateless then question would be whether the beans want to change that state:

• If the state of the collaborator bean is immutable i.e. only readable not writable then it shoud be a singleton too.

**Eğer A’nın nesnesi varsa ayrı bir yere gideceğiz, nesnesi yoksa statelessdir singleton olmalıdır.**

Bir bean’nin stateless olması demek, hiçbir fieldı yok. En iyi objeler aslında state olmayan, ve state degismeyen nesnelerdir. Yani immutable nesnelerdir.

**Bu durumda Singleton üzerinde metotlar vardır. Bütün metotların davranısı statelesstir. Yani metot çağrısı yaparsak f methodunu ve g metodunu çağırdık burada g metodunun davranısı f metodunun davranısına bağlı değildir. Mesela g metodunda bir seyler yaptık A nesnesini degisim yaptık bu durumda g metodu f metodunda ki davranısın degismesine göre davranacak diyemiyorsak bu durum stateless’tir.**

Her metot çağrısında gecilen parametreler is yapar, o parametrelerle is yaparız, cevap sadece o parametrelere bağlıdır.

**Yani calculator düsünelim toplama icin iki tane sayı geciyoruz cevabını alıyorzu, çarpma işlemi de aynı sekilde yani tamamen stateless’tir.**

Mesela calculator da argüman, operator set getlesek dönecek olan result(sonuc) yani buradaki sonuc hangi iki argümanı gectiğimiz ve hangi operatorı sectiğimiz ile ilgili 3 metodun sonucuna bağlıdır. **Yani bu durum da statefull’dur.**

**Stateless durumda metodunu cagırırken gectiğimiz parametre zaten local variable olacağı icin multithreading ortamı olsun veya olmasın sorun olmayacaktır.**

Statefull ise oradaki durumu degistirmeye calısıyor muyuz, calıstırmıyor muyuz ?

State degistiği sürece mutable bir nesneden bahsediyoruz. Sadece read-only değil write-only nesnedir.

O zaman sorumuz şuna yönelir. Aynı anda birden fazla thread var mıdır? O threadler aynı anda objenin durumunu degistirmeye kalkarsa **race conditions** olur. Bunu önlemek icin de **synchronization** **mutex loglamay**a gitmemiz lazım.

**State olmayan nesneler genelde singleton olur.**

Stateless nesne yaratmak her zaman mümkün değil.

**Mesela bir çocuk yaş değiştirdi, state’ni değiştirmemiz lazım.**

Değişmeden çoğu zaman kaçmak mümkün olmuyor, ama kaçabildiğimiz durumlarda kaçmamız gerek çünkü immutable nesneleri stateless yapabiliriz.

Mesela A objesi var türkiye deki tüm illeri ve ilçeleri tutuyor. Bunu hiçbir zaman güncellemiyoruz. Bunu bütün diğer objelere inject edebiliriz, Herkes read-only, Böyle durumlarda singleton kullanmak en iyisidir.

Eğer state değisiyorsa bu durumda birden fazla thread var mı yok mu diye bakacağız. Eğer birden fazla thread varsa

• If the collaborator bean has a mutable state then question becomes whether the environment is multi-threaded or not.

• Beans created by Spring are not thread safe.

• If the collaborator bean has mutable state then sharing it in a multithreaded environment requires synchronization to avoid race conditions and corruption in the state of the singleton.

• In this case using prototype beans could be more appropriate due to the fact that creating prototype beans can be more efficient in terms of both run-time efficiency and code complexity .

Spring’in yarattığı hiçbir nesne thread-safe değildir. En temel haliyle pojolar yaratıyor thread-safe kısmını bize bırakıyor.

A ve B threadi aynı anda bir degisiklik yapması multi-threading ortamda mümkündür. Eğer B’nin islem yapması icin A thread’nin bitmesi gerekiyorsa yani multi-threading olmadığı durumlarda bunlar aynı thread üzerinde synchron (senkron) durumda olurlar. A’nın metot çağrısı bitmeden, B metot çağrısı yapamaz. Dolayısıyla multi-threading olmayan ortamda durumu degistirmek isteseler bile **data** **corruption** önüne degisikliklerin kaybolmasının önüne gecebiliriz. A’yı nasıl kodlayacağımıza bağlı.

Multi-threading ortamda olmadığı zaman singleton kullandığımız zaman o singletonun inject edildiği objelerin yaptığı degisikliklerle alakalı ekstra islem yapmamız gerekir. Çünkü share State oldugu icin. Share state olmanın getirdiği **data** **corruption** olmadıgı halde cünkü singleton threading ortamdayız. Yine problem oluyorsa singleton da y ani complex kod oluyorsa da o zaman singleton kullanmayıp prototype yapıp herkesin kendi prototype objesine sahip olup, daha basit koda sahip olmak anlamlı.

**Eğer multi-threading ortamdaysa A, B ve C farklı threadler üzerinde ise o zaman bu 3 ayrı obje X’in üzerindeki bir durumu aynı anda degistirebilir. Her birisi islem yapar.**

• In the case of prototype all other beans would have its own copy of the

collaborator bean and will be able to change its state freely without any

need for synchronization.

• That means the state of the collaborator bean would be part of the

private state of all beans into which the collaborator bean is injected.

• On the other hand in a multi-threaded environment synchronization on

a singleton instance leads to a more complex code while using

prototype instances without any need for **synchronization** produces simpler code.

Prototype sahip olduğu zaman her bean kendi objesini alacağı icin, state multi-threading olsa bile rahatlıkla degistirebilir.

Tek bir tane oldugu zaman bu durumda **synchronization** yapmamız lazım. **Synchronization’ın getirdiği** run time ile alakalı problemler var. Kodumuz daha complex olacak. Singleton’dan vazgecelim **synchronization yapmayalım, prototype gecelim diyebiliriz.**

• Think about a calculator bean injected into many different clients that

would want the calculator bean to make calculations.

• Totally stateless calculator or unmutable state => singleton

• Statefull calculator in a single-threaded environment => singleton

• Statefull calculator in a multi-threaded environment:

• Singleton => Synchronization of the methods that change the state

• Prototype => No need for synchronization. Prototype bean is part of each client’s private state