**SINGLETON KALIBI**

* Singleton, “tek”, “tek olan”, “yegane” demektir.

**Intent** : Ensure a class only has one instance, and provide a global point of access to it.

**Amaç** : Bir sınıfın sadece bir tane nesnesinin olduğundan emin ol ve ona global bir erişim noktası sağla.

Singleton Detaylandıkca karmaşıklaşır.

**PROBLEMS**

* Singleton, yazılımlarda sıklıkla görülen, bir sınıftan sadece bir tane nesnenin bulunması gereken durumlara bir çözümdür.
* Bir sınıftan bir tane nesne olmasından kasıt ise, herkesin istediği zaman bu sınıfın bir nesnesini oluşturmaya çalışmaması, oluşturamaması, var olan tek nesneyi kullanmasıdır.
* Singleton’ın diğer bir özelliği ise global oalrak erişilebilir olmasıdır.
  + Bundan dolayı erişim amacıyla kod içinde metotlara geçerek dolaştırmaya gerek yoktur.

Böyle bir durumla çok sık karşılaşırız:

* Örneğin karmaşık bir sisteme erişimi kontrol eden sınıfın sadece bir tane nesnesinin olması istenir.
* Çünkü, bu tür nesneler trafik polisi gibi geçişi kontrol ederler, arka taraftaki sistemin vereceği hizmetlerin alınmasını düzenlerler vs.
* Ön bellek ( cache) ya da nesne /görev havuzu gibi yapılarda sistemi yöneten nesneler,
* GoF’un örnek verdiği print spooler, file system & window manager.

**Solution**

**Singleton** sınıf oluşturmanın en azından iki yolu vardır.

* **Nesne yaratmayı kontrol ederek** : Bu daha sıkıntılı bir çözüm ailesi sunar.
* **Enumeration** kullanarak: Bu daha basit ve sıkıntısı az olan çözümdür.

**Nesne Yaratma Kontrol**

Eğer singleton olacak nesne, **A** sınıfının bir nesnesi ise, aşağıdaki satırda nelerin yazılmasını kontrol ederiz?

**A a = new A();**

Bu satırda yazılmasını kontrol edebileceğimiz ya da engelleyebileceğimiz iki yer vardır:

**A ve new A()**

**A a = new A ();**

A sınıfına erişimi engelleyebilir miyiz?

Java’da hiç bir erişim niteleyici kullanmayıp **A** sınıfı package erişime sahip olursa

C#’da **A** sınıfını **internal** yapıp içinde bulunduğu **namespace** de ayrı bir assembly yapılırsa **A** sınıfına erişilemez.

C++’da sınıfa erişim kontrol edilemez.

Bu sebeple C++’da A’ya erişim engellenemez.

A sınıfına erişimi engellemek istemeyiz çünkü A her yerden erişilebilmelidir.

Bu durumda A’nın **kurucu(constructor)** çağrısının yapılmasını engelleyebiliriz.

**A a = new A();**

**Bunun için A’nın constructorlarını private yaparız.**

Bu durumda new A(); çağrısı sadece A sınıfında yapabilir.

Bu halde A sınıf şöyle olur :

Public class A{

A a = new A () ;

Private A(){}

}

* Fakat bu durumda da ortaya derin bir açmaz çıkar.
* Burada bir çelişki var :
* Bunu görmenin iki yolu vardır:
* Kolay yolu, bu sınıfın testini yazıp “a” ya ulaşmaya çalışmak,
* Bir nesnenin kendi tipinden bir nesne değişkenine (instance variable) sahip olamayacağını görmek. Imkansız durum!
* “a” bir nesne değişkenidir ve ancak ve ancak A’nın bir nesnesi yaratılırsa oluşur!
* “a” nesnesi ise ancak A sınıfında yaratılabilir.

**Bir Nokta**

Bu amaçla A’nın constructor’ını erişilebilir yapalım.

Public class A{

A a = new A () ;

A() {}

}

Public class TestA{

Public static void main (String arg){

A a = new A();

}}

Bu da yine bizim istemediğimiz durum. Bir şey değişmedi.

**BAST SINGLETON**

* Dolayısıyla “**a**” **static** olmalıdır.
* Ve “a” sarmalanarak static olan ve gelenekte ismi getInstance() olan bir metotla dışarıdan erişime açılır:

Singleton A sınıfı şöyle olur:

Public class A{

Private static A a = new A ();

Private A(){}

Public static A getInstance(){

Return a;}}

**Burada a nesnenin değil, sınıfın bir parçası olmalı.**

**Sonradan Yüklemeli Singleton ( Lazy Loading)**

* Bu çözüm en basit olandır ve üzerinde bazı iyileştirmeler yapılabilir.
* **İlk akla gelen iyileştirme static olan singleton nesnenin sınıf yüklenirken yaratılması değil aksine yaratılmasının ihtiyaç oluncaya kadar geciktirilmesidir.**
  + **Static alanlar sınıf yüklenirken yüklenirler.**
* **Bu durumda singleton nesne geç/sonradan yüklemeli ( lazy loading) hale gelir.**
* Lazy loaded singletonun problemi ise çok kanallı (multi-threaded) ortamlarda probleme yol açmasıdır.
* “null” kontrolünü aynı anda birden fazla kanalın yapması, birden fazla singleton nesnesinin oluşmasına sebep olur.

Singleton nesne oluşturmayı ulaşılamayan, **private** kurucu ile yapmak çoğu zaman problemlidir.

• Bu tür çözümler reflection ve serialization ile kırılabilir, birden fazla singleton nesne yaratılabilir.

• Java ve C#’da kurulucu **private** olsa bile reflection ile sınıfların nesneleri oluşturulabilir.

• Serialize edilmiş singleton nesne birden fazla kere okunarak belleğe farklı nesneler olarak yüklenebilir.

Singleton da lazy loading yapmasak bile reflaction ve serializable ile kırılabiliyor. Yani bu durum sıkıntılı.

**ANTİ-PATTERN**

**Singleton**’ı bir ters-kalıp (**anti-pattern**) görme eğilimi vardır.

**• Çünkü tek olan nesne erişim kolaylığından dolayı global bir değişkene dönüşmektedir.**

**• Singleton kalıbıyla hem durumu (state) hem de davranışı (behavior) global yapma imkanı vardır.**

**• Bu ise singleton nesneye ciddi bir bağımlılık oluşturmaktadır.**

• Singleton nesne, hem uygulamanın her tarafından erişilebilir durumdadır hem de muhtemelen uygulama boyunca bellekte kalmaya devam eder ve bellek sızıntısına sebep olabilir.

**Singletons Are Evil**

[**http://c2.com/cgi/wiki?SingletonsAreEvil**](http://c2.com/cgi/wiki?SingletonsAreEvil)

**Why Singletons Are Evil**

[**https://blogs.msdn.microsoft.com/scottdensmore/2004/05/25/why-singletons-are-evil/**](https://blogs.msdn.microsoft.com/scottdensmore/2004/05/25/why-singletons-are-evil/)

**Okunabilir.**

**E.Gamma On Singleton**

E. Gamma “**Design Patterns 15 Years Later: An Interview with Erich Gamma, Richard Helm, and Ralph Johnson**”’da “How would you refactor "Design Patterns?” sorusuna şöyle cevap verir:

When discussing which patterns to drop, we found that we still love them all. (Not really—I'm in favor of dropping Singleton. Its use is almost always a design smell.)

• <https://www.informit.com/articles/article.aspx?p=1404056>

**Singleton, hemen hemen her zaman, % 1,2 zamanlarda olur, ama onun dışında code smelldir, çok yaklaşmamak lazım, bağımlılık olayı singletonda problem. Organizasyonel olarak.**

**ZORLUKLAR.**

Kariyerin başında olanlar için **Singleton** kalıbı çekicidir ve sıklıkla uygunluğuna bakılmadan kullanılır.

• Geç yükleme tercih edildiğinde çok kanallı ortamlarda probleme sebep olması ve bunları önlemek için karmaşık gerçekleştirme mekanizmaları **Singleton** kalıbının uygulanmasını zorlaştırır.

**• Enumeration gerçekleştirmesiyle bu problemler ortadan kalkar.**

• Güncellenebilir duruma sahip singleton, kalıbın kullanımı daha da güçleştirebilir.

**Singleton ve Miras**

* **Singleton**’ın tüm kurucuları **private** olduğundan miras olarak devralınamazlar!
  + Yani **Singleton**’ın alt sınıfları olamaz, genişletilebilen bir yapı değildir.
  + Ama bu önemli bir kısıt değildir çünkü singletondaki davranışlar da zaten singleton yani tek olma eğilimindedir.
* Ama **Singleton** miras devralabilir.
  + Yani **Singleton**, bir arayüzden ya da sınıftan miras devralabilir, devraldığı metotlara yeni gerçekleştirmeler vererek onları ezebilir (override).

**Singleton ile Static Metotlar**

* Singleton yerine statik metotlu sınıf kullanımı da tartışılır:
  + Singleton sınıflar, interfacelerden-classlardan türetilebilirler, devraldıkları metotları ezebilirler. (override)
  + Static metotlar ezilemezler, sadece nesne metotları ezilebilirler.
* Singleton nesnenin oluşturulması ihtiyaç noktasına kadar geciktirebilir(lazy loading)
  + Static metotlu sınıflar için böyle bir durum söz konusu değildir.
    - Static alanlar sınıf belleğe yüklenince oluşturulurlar.
* Ve static kullanımı bulaşıcıdır. Aslolan nesnedir.

**SONUÇLAR**

**Singleton kalıbı şunları sağlar:**

* Tek nesnenin oluşturulmasını ve erişimi kontrol eder.
* Global değişkenlere göre daha sağlıklıdır.
* Static metotlardan daha rahat ve genişleyebilir bir yapı sunar.

**Ayrıca** Singleton sınıfının nesnelerinin sayısı değişebilir, benzer yöntemle sayısı kontrol edilebilen nesneler oluşturulabilir.

Fakat gerçekleştirme detaylarına dikkat etmek önemlidir!

**Java API’sinde pek çok singleton nesne vardır:**

**• java.lang.Runtime**

**• java.awt.Desktop**

**• java.io.Console**

Java EE API’sinde de singleton nesneler vardır:

* **Javax.ejb.Singleton**, singleton EJB oluşturan bir nottur.
* **Javax.servlet.ServletContext**
* <http://stackoverflow.com/questions/1673841/examples-of-gof-design-patterns-in-javas-core-libraries>

**Bazı kalıpların gerçekleştirilmesinde Singleton kullanılır:**

* **Abstract Factory**
* **Builder**
* **Proptotype**