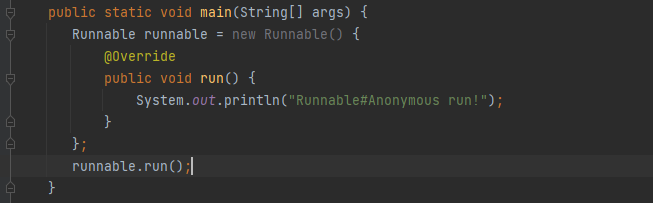
**LAMBDA EXPRESSION**

* Fonksiyonel programlamayı kolaylaştırarak kod geliştirmeyi, daha sade ve basit hale getirmeyi sağlamaktır.
* Kodumuzun daha net ve esnek olmasını sağlar.
* Lambda Expression gramerini **Functional interface**’ler ile birlikte kullanabiliriz.
* Lambda fonksiyonlarını tanımlamak için (“**->**”) operatörü kullanılmaktadir.
* **Functional interface** İçerisinde bir tane abstract metodu olan interfacelerdir. Bu yüzden **Single Abstract Method Interface** olarak da bilinirler.
* Neden tek bir soyut method olmalı? Çünkü bir lambda ifadesi, java derleyicisi tarafından bir tane soyut metodu olan interface’in nesnel karşılığına dönüştürülmektedir.
* Anonim class yazdığımız zaman bu başka bir sınıf gibi algılanıp derlenmektedir.
* Lambda ifadesiyle ise başka bir sınıf oluşturmadan normal kod gibi devam etmektedir.
* **@FunctionalInterface** anotasyonu kullanılması zorunlu değildir. Bu anotasyon sadece validasyon yapma amacıyla kullanılır. Eğer bu anotasyon eklenirse ve birden fazla abstract metot eklenmeye çalışılırsa, bu durumda compile error verecektir.
* **Functional Interfacel**er abstract metotların yanı sıra default ve static metotlara sahip olabilirler.
* Java üzerinde Runnable interfacesi buna örnektir.

**Örnek 1:** Runnable functional interface’inin eski ve yeni kullanımına bir göz atalım.

Normalde interfacelerden obje oluşturamamaktayız. Ancak interfacenin anonymous clasını oluşturabiliriz.



Yukarıdaki örneği lambda yapısında fonksiyonel olarak nasıl yazabiliriz.

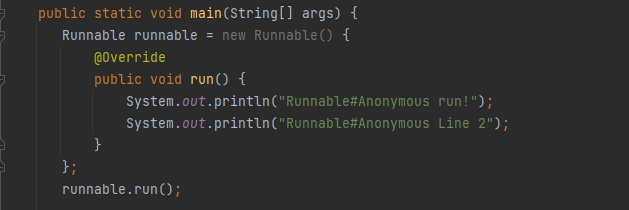
metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

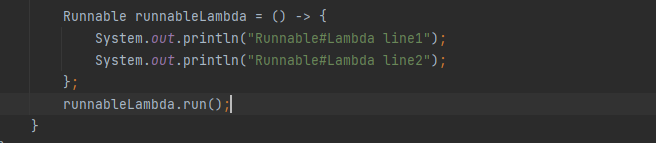
() ifadesi fonksiyonel interfacedeki metodun parametre listesidir. Eğer metot parametresizse “()” şeklinde yazılır. Bu ifadesnin kullanımı zorunludur, aksi halde derleme hatası verecektir.

**Not**: TestLambda01 sııfında bu konuda örnek yapılmıştır.

**Örnek 2:** Lambda İfadesinde birden fazla satırın kullanılması.



Yukarıdaki örneği lambda yapısında fonksiyonel olarak nasıl yazabiliriz. Bunu inceleyelim.



Birden fazla satır kullanmak istersek **“{};”** açılmalıdır.

**Parametreli Functional Interface**

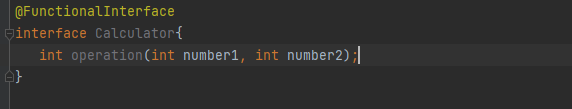
**Örnek 1:** Tek parametreli functional interface kullanımı.

metin içeren bir resim

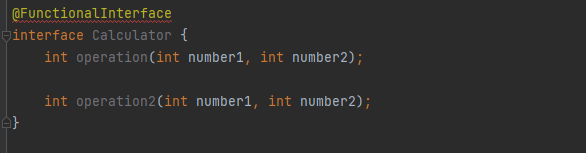
Açıklama otomatik olarak oluşturuldu

Tek parametreli kullanımlarda bu şekilde tanımlanabilmektedir. Bizim interface’imiz String bir parametre alıp void dönmektedir. Println metohuda String bir değer alıp void döndüğünden method referans şeklinde de kullanabiliriz.

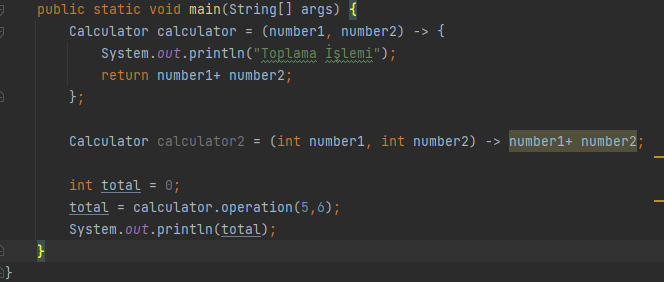
**Örnek 2:** Birden fazla parametreli functional interface kullanımı.



Eğer birden fazla abstract metot olsaydı hata alırdık, çünkü Functional interface bir tane abstract metot içerebilir.



**Calculator Functional Interface’inin kullanımı.**



Birden fazla satır kullanmak istersek **“{};”** açılmalıdır.

Tek satırlık bir işlem yapacaksak {} eklenilmeyebilir.Fakat tek satırlık bir işlemin başına return keywordunu eklemek istersek “{}” açılmalıdır aksi halde hata verecektir.

Birden fazla parametreli ise kullanım şekli;

(int number1, int number2) veya (number1,number2) şeklinde olmalıdır. Değişkenin birisine tip tanımlayıp diğerine tip tanımlamamak gibi yazımlar hata verecektir.

**Final ve Effectively Özelliği**

Local değişkeni lambda ifadesi içinde kullanabilmemiz için, değişkenimiz ya final olmalı yada bir kere atama yapılmış olmalıdır. Eğer birden fazla atama yapılırsa değişkenin effectively final özelliği bozulur. Bu nedenle compiler hatası verir ve lambda ifadesi içerisinde kullanamayız.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu kural instance değişkenler için geçerli değildir. Sadece local değişkenler için geçerlidir.

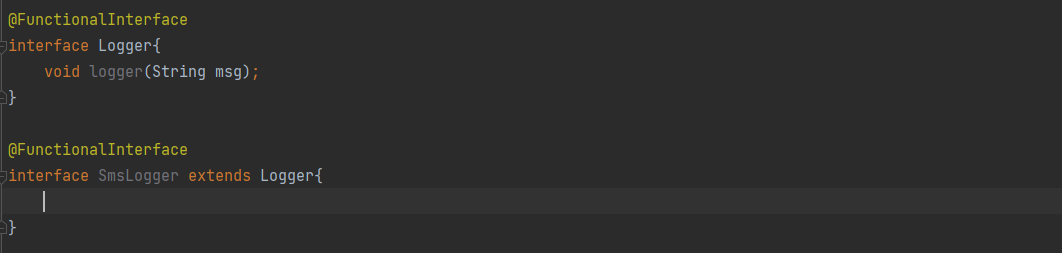
metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Functional Interface Özelliğini Bozmayan Durumlar**



Gerekli olan bir abstract method şartını sağladığı için Logger arayüzü (interface) Functional İnterfacedir.



Kendinde tanımlı olmamasına ragmen, Logger arayüzünde bir tane abstract method tanımlıdır. Gerekli olan bir abstract method şartını sağladığı için SmsLogger arayüzü (interface) Functional İnterfacedir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

DatabaseLogger arayüzü Logger arayüzündeki methodu override etmiştir.

Buda gerekli olan bir abstract method şartını bozmadığı için SmsLogger arayüzü (interface) Functional İnterfacedir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

DatabaseLogger arayüzü Logger arayüzündeki methodu override etmiş, default ve static methodlar tanımlanmıştır.

Buda gerekli olan bir abstract method şartını bozmadığı için SmsLogger arayüzü (interface) Functional İnterfacedir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

DatabaseLogger arayüzü Logger arayüzündeki methodunu ve Object sınıfındaki toString methodunu override etmiştir.

**toString**() methodu java.lang.Object sinifinda yer aldigi icin abstract metot sayisini artirmaz.

Yani Functional Interface icin gerekli olan sadece tek bir tane abstract metot sartini bozmaz.

Özel/İstisnai bir durumdur.

Buda gerekli olan bir abstract method şartını bozmadığı için SmsLogger arayüzü (interface) Functional İnterfacedir.