**Clean Code (Temiz Kod)**

Dilimize “temiz kod” olarak çevrilen clean code olgusuyla ilgili bazı bilimsel eserler mevcut olmakla birlikte akla ilk gelen kitap Robert C. Martin’in Clean Code A Handbook of Agile Software Craftsmanship kitabıdır. Temiz kod olgusunu anlayabilmek için bu eserin kısaca tanınmasında yarar görüyorum.

Clean code ile ilgili olarak bu kitap iki sebepten ötürü okunmalıdır: İlki yazılımcısınız, ikincisi daha iyi bir yazılımcı olmak istiyorsunuz. Güzel, çünkü daha iyi yazılımcılara ihtiyacımız var. Bu kitap iyi programlamayı anlatıyor. Bir sürü kodlama örneği olacak. Bu kitabı bitirdiğiniz zaman iyi kod ve kötü kod arasındaki farkı anlayabileceksiniz.

Kitap, bazı bilinen yazılımcıların temiz kod hakkındaki görüşlerine de yer vermiştir. Bunlar:

[**Bjarne Stroustrup (C++’ın mucidi)**](https://tr.wikipedia.org/wiki/Bjarne_Stroustrup): Kodumun şık ve temiz olmasını seviyorum. Kodda mantık, hataların saklanmasını zorlayacak kadar düz; bağımlılıklar (dependency) bakımı kolaylaştıracak kadar minimal olmalı. Tüm istisnai durumlar (excemptions) ele alınmalı, performans optimale yakın olmalıdır.

[**Grady Booch (Object Oriented Analysis and Design with Applications kitabının yazarı):**](https://en.wikipedia.org/wiki/Grady_Booch)Temiz kod basit ve açıktır. Temiz kod, iyi yazılmış bir düzyazı gibidir. Temiz kod, asla tasarımcının niyetini gizlemez, daha çok berrak soyutlamalarla ve düz kontrol satırlarıyla doludur.

[**Dave Thomas**](https://en.wikipedia.org/wiki/Dave_Thomas_(programmer))**(**[**OTI Labs**](https://en.wikipedia.org/wiki/Object_Technology_International)**’ın kurucusu):**Temiz kod, onu geliştiren yazılımcı dışında başka geliştiriciler tarafından da okunabilir ve iyileştirilebilir. Birim ve kabul testleri ile anlamlı isimlendirmeleri mevcuttur. Bir şeyin yapılması için tek bir yol vardır. Onun da çok az bağlılığı olup temiz bir API sağlar.

[**Michael Feathers (Working Effectively with Legacy Code kitabının yazarı):**](https://www.linkedin.com/in/michaelfeathers/)Temiz kod için bildiğim birçok özelliği sıralayabilirim; ancak bir tanesi diğer tüm özellikleri kapsıyor. O da temiz kodun her zaman ona değer veren biri tarafından yazılmış gibi görünüryor olmasıdır.

Benim temiz kod hakkındaki düşüncelerim ise, bir takım çalışmasında yazılan kodun diğer takım arkadaşları tarafından veya sonrasında geliştiren ekipler tarafından yanlış anlaşılmalara sebebiyet vermemesi gerekir. Bu nedenle kod düzgün, şık, anlaşılır ve temiz bir şekilde yazılmalıdır.

1. **İsimlendirme**

Kodumuzda değişkenlere, fonksiyonlara, dosyalara, argümanlara, sınıflara ve paketlere isimler veririz. Bu isimlerin anlaşılır olması hem bizim hem kodu okuyacak olanların daha kolay bir şekilde anlamasını sağlar ve zaman kazandırır.

Örnekler verecek olursak:

* **Anlamlı isimler kullanılmalı**

Örnek 1:

int k; //gün cinsinden geçen süre

Bu isimlendirmedeki “k” harfi bizde bir anlam ifade etmiyor. Bu harf ne gün ile alakalı ne de süre ile. Bu nedenle;

int gunCinsindenGecenSure;

gibi isimlendirme yapılırsa daha anlaşılır olacaktır.

Örnek 2:

Dikkat edilmesi gereken bir diğer husus “l” ile “1” (küçük L harfinin 1 (bir) ile benzerliği) ve “O” ile “0” (büyük o harfinin 0 (sıfır) ile benzerliği) gibi benzerliklerdir.

Bir kod bloğu inceleyecek olursak:

metin içeren bir resim

Açıklama otomatik olarak oluşturulduBu kod bloğunda gerçekListe, listeGetir, list1, item, 70 nedir? Bilinmiyor. Hepsinin bir cevabı var ama hepsi sadece yazılımcının aklındadır.

metin içeren bir resim

Açıklama otomatik olarak oluşturulduAynı kod bu şekilde yazılırsa yazılımcının aklından geçenler anlaşılmış olur.

* **Aynı anlama gelen isimler verilmemeli**

Aynı ismi aynı kapsamdaki (scope) 2 farklı objeye veremediğimiz için, birini değiştirme yoluna gideriz. Sayı eklemek yeterli olmadığında, bunu birinin bir harfini eksilterek yaparız. Ancak, isimlerin farklı olması gerekiyorsa anlamları da farklı olmalıdır. Aynı şekilde eşanlamlı sözcükler kullanmaktan da kaçınılmalıdır. Çünkü girilecek bilginin hangi değişkende tutulacağı belirsizdir.

* **Aranabilir isimler kullanılmalı**

Tek harfli değişkenler (e, a, k vb.) veya sayı sabitleri (1, 2 vb.) hem anlaşılmazlar hem de aranırken sıkıntı çıkarabilirler. Ancak döngü değişkenlerinde tek harf kullanılabilir (i, j, vb.).

1. **Koşul Kullanımı**

Kodlarda koşul, döngü veya zıplama mümkün olduğunca az kullanılmalıdır. Çünkü kişi kodu tekrar tekrar okuma gereği duyabilir. Kodlarda koşul, döngü veya zıplamalar olabildiğince doğal olmalıdır.

* **Argüman Sırası**

Örnek 1:

Hangi kod daha okunabilir?

if (yasiniz >= 18)

if (18 <= yasiniz)

Çoğu yazılımcı için ilki daha okunabilirdir. Çünkü günlük kullandığımız dile uygundur. Günlük kullandığımız dilde “Eğer en az 18 yaşındaysanız” deriz, “18 yaş, sizin yaşınızdan küçük veya eşitse” demeyiz.

* **if/else Blok Sırası**

Örnek 1:

if (a!=b){

//alternatif işlemler

} else{

//yapılması gereken işlemler…

}

if (a==b){

//yapılması gereken işlemler..

} else{

//alternatif işlemler

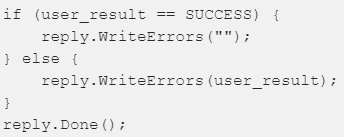
}

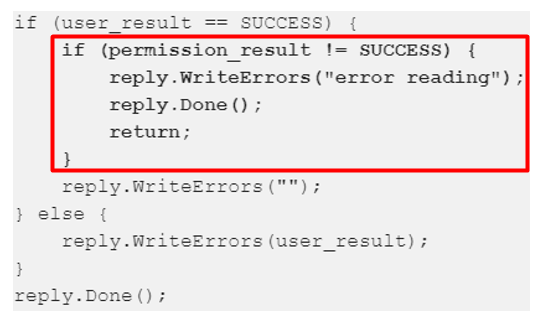
İki şekilde de yazılabilir. Ancak ilk kodu tercih etmenin bazı yararlı sebepleri olabilir;

* Negatif durum yerine pozitif durumla başa çıkmayı tercih edebilirsiniz.
* Daha basit durumla ilk olarak başa çıkmayı tercih edebilirsiniz.
* Daha ilginç veya şüpheli durumla ilk başta uğraşmayı tercih edebilirsiniz.
* **İç içe if koşulları**

Derin bir şekilde iç içe geçmiş olan kodların anlaşılması zordur.

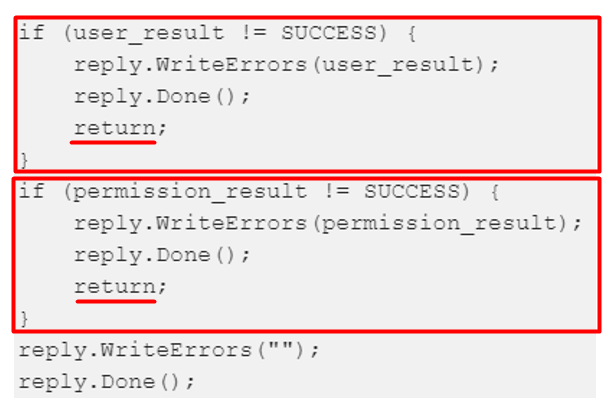
Örnek 1:

Kodun ilk hali aşağıdaki gibidir. SUCCESS durumuna yeni bir kontrol eklenilmesi isteniyor.

Yazılımcı için en kolay kodu direkt SUCCESS içine yazmasıdır. Bu da aşağıdaki gibi kodun anlaşılırlığını zorlaştıracaktır.

Sonradan eklenen kısımda kodu takip ederken, user\_result ve permissin\_result gibi iç içe yer alan değişken değerlerini takip etmek gerekiyor.

Kodu fonksiyondan erken dönme ile iyileştirebiliriz. Bunu da “hatalı durumlar” ile ilk başa çıkarak yapabiliriz.



Bu durumda yeni kodumuzda iç içe geçmiş bir if yapısı bulunmamakta ve anlaşılması daha kolaydır.

* **Kısa Devre Mantığı**

Birçok dilde, boolean operatörler kısa devre hesaplamaları yaparlar. Örneğin if(a || b) ifadesinde a true olursa, b ifadesi hesaplanmaz. if(a && b) ifadesinde ise a false olursa, b tarafı hesaplanmaz. Bu tür durumlarda neyi kontrol etmemiz gerekiyorsa önce onu yazmalıyız.

1. **Döngü Kullanımı**

* **Do/while Döngüsü Kullanmaktan Kaçınılmalı**

Do (ifadeler) while(koşul) döngüsünde kod bloğu koşula göre çalıştırılabileceği gibi çalıştırılmayabilir. Bu, kafa karıştırıcı bir durumdur. Tüm do/while döngüleri while döngüsü olarak da yazılabilir.

[Bjarne Stroustrup (C++’ın mucidi)](https://tr.wikipedia.org/wiki/Bjarne_Stroustrup) bu konu hakkında: Deneyimime göre, do-ifadeleri hatalara ve kafa karışıklığına yol açar. Yukarıda görebileceğim şartları tercih ederim. Sonuç olarak, do-ifadeleri kullanmaktan kaçınırım, demektedir.

1. **Yorum Satırları**

* **Yorumlar**

Amaç kodda kendimizi iyi ifade edemediğimiz noktaları, anlaşılması güç yerleri yorum satırlarıyla açıklamaktır. Kötü bir koda yorum yazmaktansa onları yeniden yazmak daha sağlıklı olacaktır. Aynı şekilde bir kodda yorum ne kadar az ise o kod o kadar temiz, düzgün ve anlaşılır demektir.

* **Yasal Yorumlar**

Bazen kurumsal kodlama standartları için yasal sebeplerden ötürü yorumlar yazılır. Örneğin telif hakkı (copyright) ve yazarlık (authorship) durumları gereklidir ve her kaynak dosyasının başına böyle bir yorum koymak mantıklıdır.

1. **Örnekler**

Yazımızı genel hatlarıyla özetleyecek şekilde bazı örnek kod bloklarıyla bitirmek istiyorum.

* **metin içeren bir resim

  Açıklama otomatik olarak oluşturulduÖrnek 1**

Kod bloğunu incelediğimizde ilk başta ne için yazıldığı anlaşılmıyor. İsimlendirmeler daha açıklayıcı olabilir. Örneğin okunanSayi değişkeni ne iş yapıyor? Faktöriyeli hesaplanacak olan olan sayıyı kullanıcıdan alıyor. O halde biz okunanSayi yerine faktoriyeliHesaplanacakSayi diye bir değişken tanımladığımızda daha açıklayıcı olacaktır. Aynı şekilde düşündüğümüzde digerSayi değişkeni de kafa karıştırıyor bunun yerine faktöriyel veya sayininFaktoriyeli denilebilirdi. Öte yandan if kullanımı ve döngü kullanımı konusunda kod bloğumuzda bir sıkıntı görünmemekte. Yorum satırları kodun işlevi açısından açıklayıcı olmaması bize kodun anlaşılmasında yardımcı olmuyor.

* **Örnek 2**

metin içeren bir resim

Açıklama otomatik olarak oluşturulduBu kod bloğumuzu incelediğimizde kodumuzun amacı kişinin yaşına göre koronavirüs kısıtlamalarından sorumlu olup olmadığını buluyor. Değişken gayet anlaşılır bir şekilde isimlendirilmiştir. Ancak bu kez iç içe if yapısı kodun anlaşılırlığını bozmuştur.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

İç içe if yapısından kurtulmak için kisininYasi değişkeni yasak olmayan yaş aralığını kontrol ederken, yasak olan olan yaş aralığını kontrol etmeye başlamış ve erken return kullanılarak if ifadesi 1 seviyeye düşmüş kod daha anlaşılır olmuştur.

* **Örnek 3**

metin içeren bir resim

Açıklama otomatik olarak oluşturulduBu kod bloğunun amacı girilen bir sayının kaç haneli olduğunu hesaplamasıdır. Değişkenlere verilen isimler anlaşılırdır. Lakin kodda kullanılan do-while döngüsü kodu karmaşık hale getirmiştir.

metin içeren bir resim

Açıklama otomatik olarak oluşturulduBir döngüde kontrol kısmının girişte olması kodun daha anlaşılır olmasını sağlar. Bu kodda da do-while döngüsünü while döngüsüne çevirmek kodun anlaşılırlığını arttırmış ve temiz kod prensiplerine uygun hale getirmiştir.

Çalışmamızın sonuna koyduğumuz örneklerle konun pekiştirilmesi amaçlanmıştır. Bunun faydalı bir çalışma olduğunu umuyoruz. Temiz kod prensiplerinin yazılacak büyük küçük her türlü yazılımda hem bize hem kodu inceleyen herkese fayda sağlayacağı gerçeği göz ardı edilmemelidir. İyi günler dilerim.

1. **Sonuç**

Bu çalışmada genel hatlarıyla temiz kodun ne olduğu, bir kod nasıl yazılırsa daha anlaşılır olacağı konularından söz edilmiştir. Yazılımcıların kendilerini daha rahat ifade etmeleri, projeleri daha kullanışlı ve geliştirilebilir hale getirmeleri için temiz kod prensiplerini bilmeleri, anlamaları ve kullanmalarının daha güzel sonucuna varılmıştır.

**Yararlanılan Kaynaklar**

https://medium.com/@busrauzun/clean-code-kitabindan-notlar-1-temiz-kod-derken-44e6f7a27eb0