Temiz Kod İçin 15 Öneri

Karşınıza karmaşık, düzensiz ve not içermeyen kodlar çıktığında neler hissettiğinizi çok iyi anlıyorum. Bunu önlemek için "temiz kod" önerilerini ve ipuçlarını bu yazıda bulacaksınız. Sizler de bu yazıyı paylaşarak temiz kodu teşvik edebilirsiniz.

Günümüzde sıfırdan başlanarak ticari bir yazılım geliştirilmesi, özellik, arayüz ve protokol çeşitliliğinin getirdiği kompleksite artışı ve pazara çıkış sürelerinin giderek kısalması nedeniyle neredeyse imkansızdır. Bu yüzden **mevcut kod parçalarının** doğrudan ya da değiştirilerek kullanılması bir zorunluluk haline gelmiştir.  
Yeniden kullanılacak kod parçaları kurum içinde üretilmiş olabileceği gibi, [açık kaynak kodları](https://www.karel.com.tr/bilgi/open-compute-project-acik-bilisim-projesi) ya da satın alınmış kodlar olabilir. Bu kod parçalarının **verimli ve güvenli** olarak kullanılabilmeleri için kolay anlaşılabilir ve değiştirilebilir olmaları çok önemlidir. Kurum içinde üretilmiş de olsa, kodun ilk geliştiricisinin uzun zaman önce ayrılmış olması, bir [üst düzey yöneticinin](https://www.karel.com.tr/blog/degisen-duzende-bt-yoneticileri) kodu olması durumunda, sıklıkla “koddaki şu açıklamada (// yyyy) ne demek istemiştiniz” gibi sorular sorma şansının olmaması, ya da kendi kodu da olsa altı ay sonra detayları hatırlama zorluğu, kodla yeni sahibini başbaşa bırakacaktır.

Tüm bu nedenlerle yazılımcıların aşağıdaki konuları anlaması ve **dikkat etmesi** oldukça önemlidir;

* İyi kod ve kötü kod arasındaki fark,
* İyi kod yazmak ve kötü kodu iyi koda dönüştürmek,
* İyi isimler, iyi fonksiyonlar, iyi nesneler (object) ve iyi sınıflar (class) oluşturmak,
* Maksimum okunabilirlik için kodu formatlamak,
* Kod yapısını bulanıklaştırmadan hata işlemek (error handling),
* Birim test yapmak ve test odaklı geliştirme (TDD).

Robert C. Martin’in Clean Code - A Handbook of Agile Software Craftsmanship (Temiz Kod – Çevik Yazılım İşçiliği Elkitabı) adlı eserinde**temiz kod yazmanın prensipleri**, örüntüleri (pattern) ve pratikleri anlatılmaktadır. Bu blog yazısında uzun uzun anlatamayacağım için, konuyla ilgili fikir vermek ve teşvik etmek amacıyla, kitabın isimlendirmeyle ilgili birinci bölümünü ana başlıklarıyla aşağıda özetlemeye çalıştım.

**1) Niyeti ortaya koyan isimler kullanın**  
Değişken, fonksiyon ya da sınıf ismi, neden var olduğunu, ne yaptığını, ve nasıl kullanıldığını anlatmalıdır.

*int g; // gun olarak gecen zaman*

yerine

*int gunOlarakGecenZaman;*

*int atamadanBeriGecenGun;*

*int degisikliktenBeriGecenGun;*

*int gunOlarakDosyaYasi;*

gibi isimler seçilmelidir.

**2) İsimde yanlış bilgilendirmeden kaçının**  
Bir grup hesaba gerçekten liste değilse *hesapListesi* demeyin. *hesapGrubu, birGrupHesap* ya da sadece *hesaplar* daha iyi olacaktır.

Yanıltıcı isimlendirmeye en uç örnek değişken ismi olarak küçük L ya da büyük o kullanmak olacaktır.

int a = l;

if (O == l)

a = Ol;

else

l = Ol;

Kullanılan editör ve fonta bağlı olarak küçük L birle, büyük o sıfırla karıştırılabilir.

**3) İsimleri anlamlı olarak farklılaştırın**  
İsimleri numara ekleyerek farklılaştırmak yanlış bilgi vermekten öte hiç bilgi içermez.

public static void copyChars(char a1[], char a2[]) {

for (int i = 0; i < a1.length, i++) {

a2[i] = a1[i];

}

}

Argüman olarak *kaynak* ve *hedef* kullanılırsa bu fonksiyon daha anlaşılır olacaktır.

**4) Telaffuz edilebilir isimler kullanın**  
Telaffuz edilebilir isimler kullanmak anlaşılırlığı kolaylaştıracağı gibi**takım içi iletişimi** de kolaylaştıracaktır. Telaffuz edemiyorsanız, üzerine tartışmanız da bir hayli zor olacaktır. *olsyagsdsn* (oluşturma tarihi, yıl, ay, gün, saat, dakika, saniye) değişkeninden bahsederken “Buradaki olsyag se de se ne değişkeni int olmalı” benzeri konuşmalar sıkça duyulacaktır çünkü [programlama](https://www.karel.com.tr/blog/programlama-dilleri-nedir-gelisimi-ve-tarihi) sosyal bir aktivitedir.

**5) Aranabilir isimler kullanın**  
Tek harflik isimler ve numerik sabitlerin metin içinde aranması kolay değildir. *OGRENCI\_BASINA\_EN\_COK\_SINIF*’ı bulmak *7*’yi bulmaktan kolaydır. Tek harflik isimler sadece kısa metodların içindeki lokal değişkenler için kullanılabilir.

Aşağıdaki iki kod parçasını karşılaştırınız.

for (int j = 0; j < 34; j++) {

t += (g[j]\*4)/5;

}

,

int reelGunSayisi = 4;

const int HAFTALIK\_CALISMA\_GUNU = 5;

int toplam = 0;

for (int j = 0; j < GOREV\_SAYISI; j++) {

int reelGorevGunleri = gorevKestirimi[j] \* reelGunSayisi;

int reelGorevHaftalari = (reelGunler/HAFTALIK\_CALISMA\_GUNU);

toplam += reelGorevHaftalari;

}

**6) Kodlamadan kaçının**  
İsim uzunluğu sınırlaması olan eski dillerde bu kuralı çiğnemek zorunluydu. Fortran, ilk harfin türü belirten bir kod olması kuralı nedeniyle **kodlamaya zorlayan bir dildi.** Ancak günümüzün güçlü tür kontrolü yapan modern dillerinde böyle bir sınırlama bulunmamaktadır.

*int iTelefonNumarasi;*

yerine

*int telefonNumarasi;*

kullanılırsa değişkenin *long* yapılması gerekince ismin değişmesi gerekmeyecektir.

Sınıf ve fonksiyon üye isimlerinde de *‘\_’, ‘m\_’* ya da *‘its’* benzeri kodlamalara gerek yoktur çünkü sınıf ve fonsiyonlarınız bunlara gerek duymayacak kadar kısa olmalıdır. Ayrıca üye isimlerini renklendiren bir editör kullanılmalıdır.

**7) Zihinsel eşlemelerden kaçının**  
Döngü sayaçlarına*i* ya da *j* gibi isimler verilebilir çünkü bu geleneksel bir uygulamadır. Ancak diğer çoğu durumda tek harfli bir isim kötü bir seçimdir.

Programcılar genelde akıllı insanlardır. Akıllı insallar bazen gösteriş yapmayı severler.*r*’nin *url*’nin sunucu ve şema çıkarılmış küçük harf versiyonu olduğunu hatırlayabiliyorsanız açıkça çok akıllı olmalısınız. **Akıllı programcı ile profesyonel programcı** arasındaki farklardan biri, profesyonelin, açıklığın paha biçilmez olduğunu bilmesidir.

**8) Sınıf isimleri**  
Sınıf ve obje isimleri, *musteri, WikiSayfası, Hesap, HesapCozumleyici* gibi isim ya da isim cümleleri olmalıdır. Sınıf ismi fiil olmamalıdır.

**9) Metod isimleri**  
Metod isimleri, *odemeyiErtele, sayfayiSil, sakla* gibi fiil ya da fiil cümleleri olmalıdır.

**10) Her kavram için tek kelime kullanın**  
Farklı sınıfların benzer metodları için *Sakla, Yaz* ve *Kaydet* kullanmak kafa karıştırıcıdır. Bir tanesi seçilip tüm kodda bağlı kalınmalıdır.

**11) Kelime oyunu yapmayın**  
Aynı kelimeyi farklı fikirler için kullanmak bir kelime oyunudur. *Topla*‘yı ele alalım. Bir sınıfta tuşlanan numaraların toplanması, bir diğerinde mevcut iki değerin toplanması ya da eklenmesi işlemi için kullanılması **kafa karıştırıcı olacaktır.**

**12) Çözüm alanına ait isimler kullanın**  
Kodunuzu okuyacak kişilerin programcı olacağını unutmayın. [Bilişim terimlerini](https://www.karel.com.tr/blog/7-ilginc-teknoloji-terimi-ve-hikayeleri), algoritma isimlerini, örüntü isimlerini, matematik terimlerini kullanmaktan çekinmeyin. Zorunlu olmadıkça sorun alanına ait terimler kullanmamalıyız çünkü programcıların ikide bir müşteriye gidip isimlerin ne anlama geldiğini sormalarını istemeyiz. *HesapVisitor* Visitor örüntüsüne aşina bir programcıya çok şey ifade edecekken, *HesapCetveli* fazla fikir vermeyebilir.

**13) Sorun alanına ait isimler kullanın**  
Programlayıcı jargonundan bir isim bulunamıyorsa, sorun alanından bir isim kullanılabilir. Programcı en azından bir alan uzmanı bulup ne anlama geldiğini sorabilir.

**14) Anlamlı bağlamlar ekleyin**  
*ad, soyad, cadde, sokak, numara, sehir ve postakodu* isimli değişkenleriniz olduğunu düşünelim. Topluca bakıldığında bir adres oluşturdukları anlaşılabilir. Fakat sadece kodun bir yerinde *numara*’nın tek başına kullanıldığını görürseniz ne anlarsınız? Ön ekler kullanarak bağlam ekleyebilirsiniz: *adresAd, adresSoyad, adresCadde*… Daha iyisi*Adres*isimli bir yapı ya da sınıf oluşturmak olacaktır. Böylece derleyici de bu değişkenlerin **daha büyük bir oluşumun elemanları** olduklarını bilecektir.

**15) Gereksiz bağlam eklemeyin**  
“Delüks Benzin İstasyonu” adlı hayali bir uygulamada, her sınıf isminin önüne DBI eklemek kötü bir fikirdir. Açıkcası kullandığınız araçlara karşı haksızlık yapmış olursunuz. D’ye basın ve sistemdeki tüm sınıfların bir kilometrelik listesi karşınızda.

**BONUS: Muzip olmayın**  
İsimlendirme çok akıllıca olursa sadece yazarın espri anlayışını paylaşanlar tarafından ve şaka hatırlandığı süre boyunca anlaşılabilecektir. *KutsalElBombasi* isimli bir fonsiyonun ne olduğunu kim bilebilir? Gerçekten çok muzipçe ama bu durumda *OgeleriSil* daha uygun bir isim olacaktır.