İYİ:

```
public const int aySayisi = 12;
public const int haftaSayisi = 52;
public readonly int gunSayisi = 365;
```

~~KÖTÜ~~

```
public const int Ay_Sayisi = 12;
public const int haftasayisi = 52;
public readonly int _Gun_Sayisi = 365;
```

Microsoft
Visual C# .net

TİKS ICS

Kodlama Standartları

Visual C# Kodlama Standartları

Versiyon 1.1 | 10.02.2010

Versiyon Geçmişi

Tarih	Ver	Açıklama	Yazar(lar)	Gözden Geçiren(ler)
01.01.2011	1.0	Belge oluşturuldu	-	-
10.02.2011	1.1	Belge kitap formatına çevrildi. İndeks eklendi ve hatalar düzeltildi.	Kadir Çamoğlu	Hakan Çamoğlu

Künye

Proje Koordinatörü
Kadir Çamoğlu

Editör
Kadir Çamoğlu

Kapak Tasarımı
Hakan Çamoğlu

Yazarlar

Volkan ATASEVER
Merve CEYLAN
Kadir ÇAMOĞLU
Burhan ÇETİNKAYA
Bahtiyar DİLEK
İrfan MEVSİM
Can YAPAN
Hakan YAVUZ
Yücel YILDIRIM

Gözden Geçirenler

Emir ALP
Kadir ÇAMOĞLU
Mustafa DEMİRCİOĞLU
Engin DENİZ
Mesut GÜNEŞ
Mustafa GÜR
Hürkan KARTAL
Murat KOCAVELİOĞLU
Orkan METİN
İrfan MEVSİM
Soykan ÖZÇELİK
Kadir ÖZGÜR
Levent SABAH
Selçuk TÜZEL
Kerem VARIŞ
Akın YETİM
Fatih YÜCALAR
Eren ZEDE

İçindekiler

Terimler

Terim	Türkçe	Açıklama
Namespace	İsim alanı	
Class	Sınıf	
Interface	Ara birim	
Attribute	Öz nitelik	
Enumeration	Numaralandırma	
Property	Özellik	
Event	Olay	
Constant	Sabit	
Readonly	Salt okunur	
Pascal Casing	-	İsimler BÜYÜK harfle başlar. İsmi oluşturan ifade birden fazla kelimeden oluşuyorsa, kelimelerin ilk harfleri BÜYÜK harflerle yazılır. (OgrenciKaydet, UcretHesapla, RaporYazdir)
camelCasing	-	İsimler küçük harfle başlar. İsmi oluşturan ifade birden fazla kelimeden oluşuyorsa, isimdeki ilk kelime küçük harfle başlar. Sonraki kelimeler BÜYÜK harfle başlar. (yeniOgrenciKaydet, ucretHesapla, raporYazdır)

Önsöz

Kaliteli yazılım geliştirmenin ilk adımı, yazılımı bir proje olarak ele almak ve yazılım geliştirme sürecini standartlaştırmaktır. Başarılı bir yazılım projesinin tanımı ise kısaca şöyledir: “Müşteri gereksinimlerini bütçesinde ve zamanında hatasız olarak karşılamak...”

Kaliteli bir yazılım geliştirmek için, iyi bir yazılım geliştirme yönteminiz olmalı ve bu yöntem test pratikleri, kodlama pratikleri, kodlama standartları gibi birçok dokümanla desteklenmelidir. Böylece ekip halinde yapılan çalışmalarda, ekip üyeleri birbirlerinin kodunu daha rahat okuyabilir ve anlayabilir. Tek başınıza çalışıyor olsanız bile, üzerinden zaman geçtikten sonra kodu daha rahat okuyabilmeniz için kendinize has isimlendirme standardınız olması gerekir.

Peki, “Kodlama Standartları” neleri içerir?

Kelimenin tam anlamıyla “Kodlama Standartları” olarak baktığımızda, kastettiğimiz şey, değişkenlere, sınıflara, metotlara, parametrelere, vb. öğelere nasıl isim vereceğimiz ve onları nerede nasıl yazacağımızla ilgilidir. Ama kaliteli bir yazılım geliştirmek için bundan daha fazlasına ihtiyacımız olacaktır. Kullanıcı ara yüzlerinin nasıl geliştirileceği, hatanın nasıl ele alınıp işleneceği, operasyonların kayıt günlüğüne aktarılması, veri tabanı işlemlerinin nasıl olacağı, listelerin nasıl doldurulacağı gibi, tüm uygulamalarda sürekli tekrarlanan işlerin nasıl yapılacağını belirlemek de bence kapsam içine alınabilir. Bu durumda epey detaylı bir dokümana ulaşırız ve bu tabii ki biraz da emek gerektirir. Ama buna fazlasıyla değecektir.

Tüm programlama dilleri için geçerli, tüm projelerde uygulanabilecek ve her takıma uygun tek bir yazılım geliştirme standartları dokümanı hazırlamak pek mümkün değildir. Ancak içinde bulunulan kuruma/projeye/ekibe uygun bir şeyler yapmak için genel bir şablon oluşturmak hiç de zor olmayacaktır.

İşte bu çalışmanın amacı, size başlangıç şablonunu oluşturacak temel bir doküman sağlamaktır. “C# Kodlama Standartları”, “Visual Basic Kodlama Standartları” ve “TSQL Kodlama Standartları” dokümanlarını da kapsayan bir dizi standart dokümanının ilkidir. Eğer bu belgenin faydalı olduğuna inanır, yeterince iyi geri bildirim alırsak, dizinin sonraki dokümanlarını 2011 yılının ilk yarısında oluşturmayı planlıyoruz.

Bu proje fikrini Facebook’taki “Tek Kişilik Yazılımevi” grubuyla paylaştığımda beklediğimden çok daha fazla destek aldım. İçerik geliştirmede, gözden geçirmelerde, projenin ilerleme aşamasında fikirleriyle ve emekleriyle bizi destekleyen herkese çok çok teşekkür ediyorum. Özellikle içerik geliştiren arkadaşlara ayrıca minnetlerimi iletiyorum.

Umarım geç de olsa oluşturduğumuz bu doküman işinize yarar.

Kadir Çamoğlu
Aralık 2010

Giriş

Amaç

Bu dokümanı geliştirmekteki amacımız, C# programlama diliyle geliştirilecek projelerde kullanılmak üzere, kodlama standartları için, bir "temel", fikir verecek bir "örnek" oluşturmaktır.

Kapsam

Bu doküman, C# Kodlama Standartları belgesinin ilk sürümüdür. Bu nedenle isimlendirme, biçimlendirme ve temel kodlama yapıları gibi en temel konulara değinilmiştir.

Kullanım Şartları

Bu doküman bir grup çalışmasının ürünüdür. Belgenin sahipleri teknik olarak künyede yazılı olan (gözden geçirenler hariç) isimlerdir. Belgeyi, kaynak belirtmek koşuluyla tamamen ya da kısmen kullanabilirsiniz. Kopyalayabilir, çoğaltabilir ve dağıtabilirsiniz.

Kullanım Önerisi

Bu belgeyi geliştireceğiniz projelerde olduğu gibi kullanmanız mümkün olabileceği gibi, kendi standartlarınızı oluşturmak için temel almak üzere de kullanabilirsiniz. Özellikle daha önceden edindiğiniz yazılım geliştirme alışkanlıklarınız varsa, bunları dokümandakilerle karşılaştırarak uygun gördüğünüzü tercih edebilirsiniz.

Önemli Açıklama

Bu dokümanda belirtilen hiçbir kodlama standardı, ulusal ya da uluslararası genel bir standardı ifade etmez. Bu dokümanda açıklanan standartlar, içeriği geliştirenlerin ve proje koordinatörünün tecrübeleriyle oluşturdukları, kendi projelerinde uyguladıkları kodlama yaklaşımlarıdır. Bu dokümanda yazılanları C# programlama dilinin, ilişkili teknolojilerin ya da yazılım mühendisliğinin temel kuralları olarak görmeyin. Dokümanı geliştirenlerin hiçbir şekilde en doğru ve en iyi yazılım kodlama standardını oluşturmak gibi bir iddiası yoktur. Buradaki amaç bir öneri ve/veya örnek oluşturmaktır.

Geri Bildirim

Bu proje gönüllük esasına dayalı olarak, geniş bir katılımcı kitlesiyle hazırlanmıştır. Bu dokümanın olgunlaşması ve daha kaliteli hale gelmesi için, dokümanı inceleyen, kullanan herkesin geri bildirimi bizim için çok değerli olacaktır. Lütfen dokümanla ilgili hataları, önerilerinizi ve her türlü geri bildiriminizi aşağıdaki web ve eposta adresleri üzerinden bizimle paylaşın.

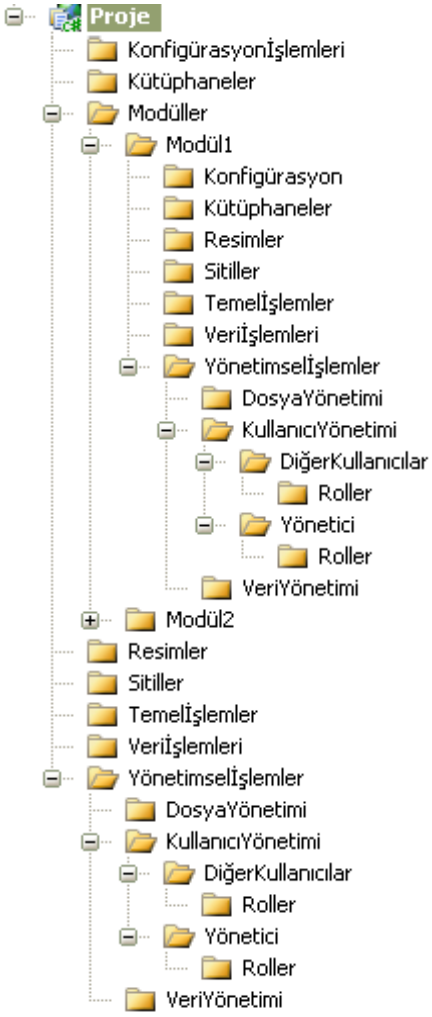
Eposta: tiksprojesi@gmail.com
http://tiksprojesi.blogspot.com/
http://www.facebook.com/#!/group.php?gid=29006737231
http://www.chip.com.tr/blog/kadircamoglu/

1. Dosya Organizasyonu

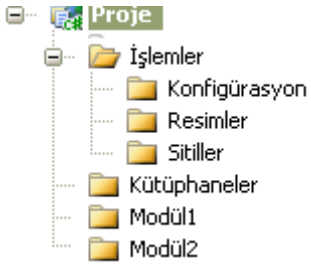
1.1 Klasör Yapısı

Kullanım amaçlarına yönelik olarak, kaynak kodlarını bir arada tutacak şekilde klasör yapısı oluşturun. Klasörün amacına yönelik isimlendirme yapın. Böylelikle hangi dosyanın nerede olduğu kolayca bulunabilir.

İYİ:



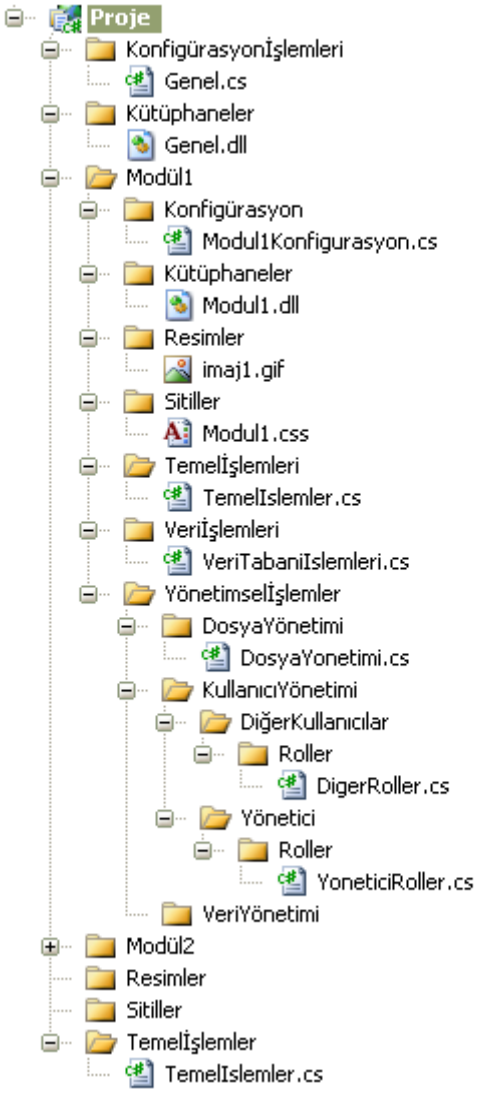
~~KÖTÜ:~~



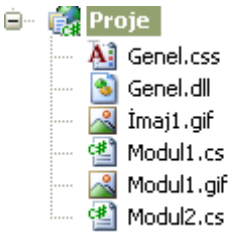
1.2 Kaynak Kod Dosyaları

Kaynak kod dosyalarını klasör yapısına uygun olarak, amaçlarına uygun yerde oluşturun.

İYİ:



~~KÖTÜ:~~



2. Biçimlendirme

2.1 Sınıf Yapısı

Sınıfların isimlendirilmesinde “Pascal Casing” kullanın.
İsimlendirmede, ismin sonuna Class veya Sınıf kelimesi eklemeyin.
Sınıf Erişim belirteçlerini (private, public, internal) mutlaka belirtin.
Her sınıf için mutlaka en az bir yapıcı metot (Constructor) oluşturun.
Parçalı (partial) sınıflar kullandığınızda, her bir parça içerisinde, diğer parçaların yerini belirtin.
Yapıcı metodumuzun içinde parametrelerinizi eşleştirirken mutlaka “this” ekini kullanın.

İYİ:

```
/// <summary>
/// Hastanın kullandığı ilaç bilgisi
/// </summary>
private class HastaIlac
{

      public HastaIlac(string ilacAdi, string ilacBarkod)
      {
      }
}
```

~~KÖTÜ~~

```
/// <summary>
/// Hasta Kayıt işlemleri
/// </summary>
public cass HastaIslemleriSinifi
{
      public HastaIslemleriSinifi()
      {
      }
}
```

Sınıfları oluştururken girintileri doğru şekilde uygulamaya dikkat edin.

İYİ:

```
/// <summary>
/// Hastanın kullandığı ilaç bilgisi
/// </summary>
private class HastaIlac
{
      private string ilacAdi;
      private string ilacBarkod;

      public HastaIlac(string ilacAdi, string ilacBarkod)
      {
            this.ilacAdi = ilacAdi;
            this.ilacBarkod = ilacBarkod;
      }
}
```

~~KÖTÜ:~~

```
private class HastaİlaçSınıfı
{
private string ilacAdi;
private string ilacBarkod;

public HastaİlaçSınıfı(string ilacAdi, string ilacBarkod)
{
ilacAdi = ilacAdi;
ilacBarkod = ilacBarkod;
}
}
```

2.2 Girintileme ve Kapsamı Belirleme

Girinti kullanırken dikkatli olun. Girinti (sekme/tab) ve boşlukları uygun şekilde kullanın. Kod ya da açıklama yazarken boşluk kullanmayın. İsim alanı, sınıf, metot, koşul/döngü/istisnai durum yönetimi bloklarının her biri için ayrıca girintileme yapın.

İYİ:

```
/// <summary>
/// Hasta Kayıt işlemleri
/// </summary>
public class HastaIslemleri
{
      //Yapıcı Metodumuz...
      public HastaIslemleri()
      {

      }

      //Hastamızın hastane işleri
}
```

~~KÖTÜ:~~

```
namespace KodlamaStandart.Hastane.Hastalar
{
/// <summary>
/// Hasta Kayıt işlemleri
/// </summary>
public class HastaIslemleri
{
//Yapıcı Metodumuz...
public HastaIslemleri ()
{

}

//Hastamızın hastane işleri
public void hastaKayit(string tcKimlikNo)
{

}
}
}
```

2.3 Boşluklar

Birden fazla değişken tanımlarken, değişken adları arasında, virgülden sonra boşluk kullanın. Blok deyimler içinde, değişkenler ve operatörler arasında boşluk kullanın. Fonksiyon parametreleri arasında, virgülden sonra boşluk kullanın.

İYİ:

```
int yas, boy, kilo;
…
if ( yas == ahmet.Yas )
…
private void YasHesapla( DateTime dogumGunu,
                         DateTime hesaplanacakTarih )
```

~~KÖTÜ:~~

```
int yas,boy,kilo;
…
if(yas==ahmet.yas)
…
private void YasHesapla(DateTime dogumGunu,DateTime
                        HesaplanacakTarih)
```

3. Açıklamalar

3.1 Blok Açıklamalar

Blok açıklamalarını fonksiyon, modül ya da proje gibi kodların başına daha sonraki kullanımlara gerekli olabilecek açıklamaları girmek için kullanın. Açıklamaları çok detaya inmeden, genel hatlarıyla kod bloğunun kullanımını ve gerekli olabilecek diğer bilgileri kullanarak oluşturun.

İYİ:

```
// Satış fiyatı hesaplaması 25.10.2011 İrfan Mevsim
//
// Fonksiyon birim fiyatı ve KDV orani olarak gönderilen
// bilgilerin satış fiyatını geri döndürür.
//
// Örnek Kullanım
// Dim SatisFiyati as Double = SatisHesapla(1000,18)
//
// Dönüş Değeri --> 1018.0 (double)
//
private double SatisHesapla(int birimFiyati, int kdvOrani)
{
      return birimFiyati + (birimFiyati * kdvOrani) / 100;
}
```

~~KÖTÜ:~~

```
// Satış fiyatı hesaplaması  (yetersiz açıklama)
private double SatisHesapla(int birimFiyati, int kdvOrani)
{
      return birimFiyati + (birimFiyati * kdvOrani) / 100;
}
```

~~KÖTÜ:~~

```
private double SatisHesapla(int birimFiyati, int kdvOrani)
{
// Dönüş Değeri --> 1018.0 (yanlış ve yetersiz açıklama)
      return birimFiyati + (birimFiyati * kdvOrani) / 100;
}
```

3.2 Tek Satırlık Açıklamalar

Tek satırlık açıklamaları alt kısmında bulunan kod bloğunu açıklar nitelikte, belirli işlemlerin gerçekleştiği kod bloklarının başına yazın.

İYİ:

```
// Satış fiyatı hesaplaması
int birimFiyati = 1000;
int kdvOrani = 18;
int miktar = 3;
double satisFiyati = miktar * (birimFiyati + (birimFiyati *
                                kdvOrani) / 100);
```

~~KÖTÜ:~~

```
// Birim fiyat tanımlaması
int BirimFiyat = 1000;
int kdvOrani = 18;
int miktar = 3;
double satisFiyati = miktar * (birimFiyati + (birimFiyati *
                                kdvOrani) / 100);
```

3.3 Satır İçi Açıklamalar

Satır içi açıklamaları yaparken, sadece bulunduğu satırdaki ifadeyi açıklamak için ve sadece o satıra ait olan bilgileri girin.

İYİ:

```
    string kullaniciAdi     = "Ahmet";  // Kullanıcının Adı
    string kullaniciSoyadi  = "Kozdağ"; // Kullanıcının Soyadı
    string kullaniciSicilNo = "D201056";//Kullanıcının Sicil No
```

~~KÖTÜ:~~

```
    string kullaniciAdi = "Ahmet";      // Kullanıcının Adı
    string kullaniciSoyadi = "Kozdağ";
    string kullaniciSicilNo = "D201056";
```

~~KÖTÜ:~~

```
    string kullaniciAdi = "Ahmet" // Kullanıcının Bilgileri
    string kullaniciSoyadi = "Kozdağ";
    string kullaniciSicilNo = "D201056";
```

3.4 Dokümantasyon Amaçlı Açıklamalar

Dokümantasyon amaçlı açıklamaları; yazdığınız herhangi bir sınıf ya da fonksiyonun açıklamasını yapmak amacıyla kullanın. Açıklama yaptığınız metoda uygun olarak özet (summary), parametre ve dönüş değerleri için açıklamalar yazın.

İYİ:

```
/// <summary>
/// Basit bir Fonksiyon.</summary>
/// <param name="yazi">
/// Gelen yaziyi txtMetin.text 'e eşitler</param>
public void MetinYaz(string yazi)
{
      txtMetin.Text = yazi;
}


/// <summary>
/// Başka bir gereksiz fonksiyon </summary>
/// <returns>
/// geri dönen değer 0 olmalı.</returns>
public int OrnekFonksiyon()
{
      return 0;
}
```

3.5 TODO Açıklamaları

TODO: açıklamalarını projeyi geliştirirken aklınıza gelen ama daha sonra yapılması gereken işlemler için kullanın. Böylece projenin hangi aşamasında hangi değişikleri yapmanız gerektiğini Projenizi derlemeden önce, TODO: kelimesini kaynak kodlarınız içerisinde arayarak rahatlıkla bulabilir ve yapmanız gerekeni hatırlayabilirsiniz.

İYİ:

```
   //TODO: Hesaplamadan önce KDV oranlarının güncelliğini
kontrolet
   double satisFiyati = birimFiyati +
                        (birimFiyati * kdvOrani) / 100;
```

~~KÖTÜ:~~

```
   //TODO: Satış fiyatı hesaplanması
   double satisFiyati = birimFiyati +
                        (birimFiyati * kdvOrani) / 100;
```

4. İsimlendirme Kuralları

4.1 Büyük/Küçük Harf Kullanımı

Özellik ve değişken gibi öğeleri isimlendirirken kelimelerin ilk harflerini küçük diğer harflerini büyük kullanın. Sınıf ve metotlarda ise ilk harfleri büyük kullanın.

İYİ:

```
    int toplamaYap;
    …
    string cikarmaYap;
    …
    public int ToplamaIslemi(int ilkSayi, int ikinciSayi)
    …
    class Kisiler
```

~~KÖTÜ:~~

```
    int TOPLAMAYAP;
    …
    string CIKARMAyap;
    …
    public int toplamaisleMI(int ilkSAYI, int ikinciSAYI)
    …
    class KISILER
```

4.2 Pascal/Camel Casing

İsimlendirme yaparken Pascal/Camel isimlendirmeyi doğru yapmak için aşağıdaki tabloyu kullanın. Metotlarda, alanlarda (field) ve özelliklerde (properties) public, private, internal da olsa Pascal Casing kullanın.

Proje dosyası	Pascal Casing	HastaTakibi.csproj
Kaynak dosyası	Pascal Casing	HTLogo.jpg
Sınıf (class)	Pascal Casing	Kisi
Yapı (struct)	Pascal Casing	KisiListeBilgisi
Arabirim (interface)	Pascal Casing	IKisiselBilgiler
Metot	Pascal Casing	KisiEkle
Özellik (Property)	Pascal Casing	Adi, Soyadi
Alan (public field)	Pascal Casing	TCKimlikNo
Alan (diğer Field)	Camel Casing	dogumTarihi
Numaralandırma (Enum)	Pascal Casing	KayitDurumu
Delegate (public)	Pascal Casing	KayitSilindi
Delegate (diğer)	Camel Casing	kayitIsleniyor
Değişken	Camel Casing	vergiNumarasi
Parametre	Camel Casing	ad, soyad, yas
Olay (event)	Camel_Casing	btnKaydet_Click

4.3 C# Diline Özel Ayrılmış Kelimeler

Aşağıdaki kelimeler C# programlama diline özeldir. **Bu kelimeleri farklı büyük/küçük harf kombinasyonlarında da olsa kullanmayın.**

abstract	event	new	struct
as	explicit	null	switch
base	extern	object	this
bool	false	operator	throw
break	finally	out	true
byte	fixed	override	try
case	float	params	typeof
catch	for	private	uint
char	foreach	protected	ulong
checked	goto	public	unchecked
class	if	readonly	unsafe
const	implicit	ref	ushort
continue	in	return	using
decimal	int	sbyte	virtual
default	interface	sealed	volatile
delegate	internal	short	void
do	is	sizeof	while
double	lock	stackalloc	
else	long	static	
enum	namespace	string	

4.4 Büyük/Küçük Harf Hassasiyeti

Türkçe karakterler farklı algılandığından kodlama içerisinde Türkçe karakterler kullanmayın. Küçük i ve bu karakteri büyük yazarken I kullanın. Ç yerine c, ş yerine s, ğ yerine g, ü yerine u, ö yerine o tercih edin.

İYİ:

```
int kacisVeIlk;
string ogrenmeVeEgitim;
```

~~KÖTÜ:~~

```
int KaçışveİIk;
string ÖğrenmeVeEğitim;
```

Fonksiyonlarda parametre kullanırken verilen tek karakterli isimleri büyük veya küçük versiyonlarını yazarak kullanmayın.

İYİ:

```
    public int ToplamaIslemi(int istenenSayi, int verilenSayi)
```

~~KÖTÜ:~~

```
    public int ToplamaIslemi(int b, int B)
```

Aynı isimli fakat harflerinin büyüklüğü küçüklüğü farklı değişkenler tanımlamayın.

İYİ:

```
    int denemeSayisi;
    int eldeSayisi;
```

~~KÖTÜ:~~

```
    int deneme;
    int DENEME;
```

C# diline özel olan ayrılmış kelimeleri büyük/küçük harf durumunu değiştirerek de olsa kullanmayın.

İYİ:

```
    Point degisken;
    int digerDegisken;
```

~~KÖTÜ:~~

```
    Point p;
    int POINT;
```

Farklı büyük/küçük harf hassasiyetlerinde, aynı isimli metotlar yazmayın.

İYİ:

```
    private void ToplamaHesabi()
    {
    }

    private void CikarmaHesabi()
    {
    }
```

~~KÖTÜ:~~

```
    private void hesapla()
    {
    }

    private void Hesapla()
    {
    }
```

4.5 Kısaltmalar

Sınıf, özellik, metot, değişken gibi öğelere isim verirken çok zorunlu olmadıkça kısaltma kullanmayın.

İYİ:

```
int kullaniciAdi;
bool yetkiliMi;
 …
private void KullaniciKaydet(){}
```

~~KÖTÜ:~~

```
int kAd;
bool ytkl;
…
private void KullKyd(){}
```

Eğer aynı isme sahip aynı türden iki veya daha fazla öğe kullanılacaksa bunları numaralandırmayı tercih edin.

İYİ:

```
int ogrenci1;
int ogrenci2;
```

~~KÖTÜ:~~

```
int ogrenciIlk;
int ogrenciIkinci;
```

4.6 Kelime Seçimi

Kolay okunabilir, tanımlayıcı isimleri seçin.

İYİ:

```
public int UrunAdi { get; set; }
```

~~KÖTÜ:~~

```
public int AdiUrun { get; set; }
```

Tanımlayıcı isimlerinizi kısa tutmak için onları anlamsızlaştırmayın.

İYİ:

```
public int KullaniciAdi { get; set; }
```

~~KÖTÜ:~~

```
public int kAdi { get; set; }
```

Tanımlayıcılarınızı isimlendirirken hiçbir zaman tire "-", alttan tire "_" veya alfa-numerik olmayan karakterler "!@#$%^&*()" kullanmayın.

İYİ:

```
public DateTime DogumTarih { get; set; }
```

~~KÖTÜ:~~

```
public DateTime Dogum&Tarih;
public DateTime Dogum_Tarih;
```

Tanımlayıcılarınızı hiçbir zaman Macar Notasyonu - Hungarian Notation- ile isimlendirmeyin.

~~KÖTÜ:~~

```
public int u32Identifier;
public bool bBusy;
public char chInitial;
public double dbPi;
public string[] rgStudents = new string[5];
public Form1 hwndForm;
public int u32Identifier;
public bool bBusy;
public char chInitial;
```

4.7 "İsim Alanı" İsimlendirme

İsim alanı adlandırırken, "Pascal Casing" tipi isimlendirme kullanın. Büyük harfle başlayıp her bir kelimenin ilk harfini büyük harf kullanarak isim verin. Kelimeler arasında "_" veya "-" benzeri ayraçlar kullanmayın. İsim alanı adlandırırken, proje/assembly adı ile aynı adı kullanın. Kullandığınız dizinlerin adlarını, isim alanı adlarında da kullanın.

İYİ:

```
KodlamaStandartlari.csproj
namespace KodlamaStandartlari

KodlamaStandartlari.Models.dll
namespace KodlamaStandartlari.Models

KodlamaStandartlari.csproj projesinde Models dizini
namespace KodlamaStandartlari.Models
```

~~KÖTÜ:~~

```
KodlamaStandartlari.csproj
namespace Kodlama

KodlamaStandartlari.Models.dll
namespace KodlamaStandartlari.Modeller

KodlamaStandartlari.csproj projesinde Models dizini
namespace KodlamaStandartlari.Modeller
```

4.8 "Sınıf" İsimlendirme

Sınıflara ad verirken, Pascal tipi isimlendirme kullanın. Büyük harfle başlayıp her bir kelimenin ilk harfini büyük harf kullanarak isim verin. Kelimeler arasında "_" veya "-" benzeri ayraçlar kullanmayın. Sınıflara ad verirken, isim veya isim öbeği kullanın. Sınıflara adlarında, "Sınıf", "Class" veya "C" gibi ifadeler kullanmayın.

İYİ:

```
public class Araba
…
internal class Ucurtma
…
private class UcakMotoru
```

~~KÖTÜ:~~

```
public class Yap
…
internal class ClassUcurtma
…
internal class CucurtmaSinifi
…
private class UcakmotoruClass
```

4.9 "Ara Birim" İsimlendirme

Ara birimlerinizi uygulayacağınız sınıfların isimleriyle birlikte başlarına "I" harfi getirerek yazın. Ara biriminizi sınıfınıza uygularken metotlarınızı #region - #endregion ön işlemci komutları arasına yazın. Ara biriminizi isimlendirirken her iki kelimenin ilk harfleri büyük olacak şekilde isimlendirin.(PascalCasing)

İYİ:

```
public interface ITakip
{
     public int StokTakip(int urunAdet);
}

public class Takip :ITakip
{
     #region ITakip Members
     public int StokTakip(int urunAdet)
     {
      this.urunAdet = urunAdet;
     }
     #endregion
}
```

~~KÖTÜ:~~

```
public interface StokInterface
{
     public int StokTakip(int urunAdet);
}

public class Takip :StokInterface
{
     #region StokInterface Members
     public int StokTakip(int urunAdet)
     {
      this.urunAdet = urunAdet;
     }
     #endregion
}
```

4.10 "Öznitelik" İsimlendirme

Eğer sınıfınıza birden fazla öznitelik ekleyecekseniz her özniteliği yeni bir satıra yazın. Sadece ilişkili öznitelikleriniz aynı satıra yazın. Kendi geliştirdiğiniz "öznitelik" sınıflarınızı isimlendirirken, sınıf isminizin sonunda "Attribute" kelimesini getirin.

İYİ:

```
[Attribute1]
[Attribute2]
[Attribute3, RelatedAttribute3]
public class Takip
{}
```

~~KÖTÜ:~~

```
[Attribute1, Attribute2, Attribute3]
public class Takip
{}
```

4.11 "Numaralandırma" İsimlendirme

Numaralandırmalarınızı isimlendirirken "Pascal Casing" isimlendirme kuralını dikkate alın. Numaralandırma isimlerinizin sonuna "enum" yazmayın. Numaralandırmalarınız için tekil isimler seçin.

İYİ:

```
public enum OnemDerecesi
{
     Yuksek, Orta, Dusuk
};
```

~~KÖTÜ:~~

```
public enum OnemDereceleriEnum
{
     Yuksek, Orta, Dusuk
};
```

4.12 "Sabit" ve "Salt Okunur" Alan İsimlendirme

"Sabit" ve "Salt Okunur" alanlar isimlendirilirken; isim, isim tamlaması ve (çok uzunsa) anlamlı kısaltmaları; her kelimenin ilk harfi büyük olacak şekilde kullanın. (PascalCasing)

İYİ:

```
public const int AySayisi = 12;
public const int HaftaSayisi = 52;
public readonly int GunSayisi = 365;
```

~~KÖTÜ:~~

```
public const int Ay_Sayisi = 12;
public const int haftasayisi = 52;
public readonly int _Gun_Sayisi = 365;
```

4.13 "Statik" Alan İsimlendirme

"Statik" alanları isimlendirilirken; isim, isim tamlaması ve (çok uzunsa) anlamlı kısaltmaları; her kelimenin ilk harfi büyük olacak şekilde kullanın. (PascalCasing)

İYİ:

```
public static double IndirimliFiyat;
public static int ToplamSaat;
```

~~KÖTÜ:~~

```
public static double fyt;
public static double _fiyat;
public static int  toplam_saat;
```

4.14 Parametre İsimlendirme

Parametre isimlendirilirken; içeriğine göre anlamlı olarak, isim, isim tamlaması ve (çok uzunsa) anlamlı kısaltmaları; birinci kelimenin ilk harfi küçük bunu takip eden kelimelerin baş harfleri büyük olacak şekilde kullanın. (camelCasing)

İYİ:

```
private void KullaniciKaydet(string kullaniciAdi, string parola) {}
```

~~KÖTÜ:~~

```
private void KullaniciKaydet(string kul_adi, string prl ) {}
```

4.15 Metot İsimlendirme

Metotlarınızı isimlendirirken “Pascal” isimlendirme kuralını kullanın.(Pascal Casing)
Metot parametreleriniz için “Camel” isimlendirme kuralını kullanın.(Camel Casing)
Metotlarınızı fiille veya fiil cümleleriyle isimlendirin.

İYİ:

```
public int Topla(int birinciSayi, int ikinciSayi)
{
     return birinciSayi + ikinciSayi;

}
```

~~KÖTÜ:~~

```
public int Toplamaislemi(int BirinciSayi, int IkinciSayi)
{
     return BirinciSayi + IkinciSayi;
}
```

4.16 Özellik İsimlendirme

Özelliklerinizi isimlendirirken Pascal isimlendirme kuralını kullanın.(Pascal Casing)
Özelliklerinizi isim ve isim tamlamalarıyla isimlendirin. Özelliklerinizin tiplerine uygun isimlendirme yapın. Özelliklerinizi isimlendirirken başlarına Get veya Set getirmekten sakının.

İYİ:

```
public int IsimSoyisim { get; set; }
public DateTime OdemeTarihi { get; set; }
//OnemDerecesi Enum bir tipe sahiptir.
public OnemDerecesi OnemDerecesi { get; set; }
```

~~KÖTÜ:~~

```
public string Getisim { get; set; }
public DateTime OdemeTarihiniAl { get; set; }
public OnemDerecesi SetOnemDerecesi { get; set; }
```

4.17 Olay İsimlendirme

Olay(Event) işleyici isimlendirmede “on” ifadesini kullanmayın.

İYİ:

```
event EventHandler ciftTikla;
```

~~KÖTÜ:~~

```
event EventHandler onCiftTikla;
```

Tek bir “event” isimlendiriyorsanız geçmiş zamanda fiiller kullanın.

İYİ:

```
event EventHandler mesajYollandi;
event EventHandler ciftTiklandi;
```

~~KÖTÜ:~~

```
event EventHandler mesajYolla;
event EventHandler ciftTikla;
```

4.18 Değişken İsimlendirme ve İlk Değer Atama

Vereceğimiz isimler Türkçe olmalıdır. Tanımladığımız isimler olmalı ancak anlamını kaybetmemelidir.

İYİ:

```
class ResimBoyutlandirici
{
      string resimYolu = "c:";
      string cozunurluk = "1024x768";
}
```

~~KÖTÜ:~~

```
class WindowsFormsApplication1
{
      string picPath = "c:";
      string @int = "1024x768";
}
```

4.19 Kontrol İsimlendirme

Kontrol İsimlendirirken, kontrollerin önüne, ilgili kontrolü belirtecek bir kısaltma kullanın.

İYİ:

```
private void btnBaslat_Click(object sender, EventArgs e)
{
}

private void lblAciklama_Click(object sender, EventArgs e)
{
}
```

~~KÖTÜ:~~

```
private void Baslat_Click(object sender, EventArgs e)
{
}

private void Aciklama_Click(object sender, EventArgs e)
{
}
```

Kontrol isimlendirme için aşağıdaki tabloyu kullanın.

Kontrol	**Kısaltması**
Checkbox	chk
RadioButton	rdb
Label	lbl
Textbox	txt
DataGrid	dtg
Button	btn
ImageButton	İmb
Image	img
PlaceHolder	phd
Table	tbl
CheckBoxList	cbl
RadioButtonList	rbl
Radio(Option)Button	opt
Hyperlink	lnk
DropDownList	ddl
ListBox	lst
DataList	dtl
Repeater	rpt
Panel	pnl
Validators	val
ComboBox	cmb
DropDownList	ddl
GridView	gvw
hScrollBar	hsc
vScrollBar	vsc
Picture	pic
Container	ctr
DataColumn	dcol
DataGridDateTimePickerColumn	dgdtpc
DataGridTableStyle	dgts
DataGridTextBoxColumn	dgtbc
Dialog	dialog
DialogResult	dr
GroupBox	gbx
ImageList	iml
MarshallByRefObject	rmt
Mainmenu	mm
MenuItem	mni
NumericUpDown	nud
PictureBox	pbx
RadioButton	rbtn
SqlCommand	sqlcom
SqlCommandBuilder	sqlcomb
SqlConnection	sqlcon
SqlDataAdapter	sqlda
StatusBar	stb
TabControl	tabctrl
TabPage	tabpage
ToolBar	tbr
ToolBarButton	tbb

5. Deyimler

5.1 Basit Deyimler

Basit deyimleri yazarken her bir satıra sadece bir deyim yazın. Deyimleri açık ve okunur şekilde kodlayın.

İYİ:

```
int sayi1 = 5;
int sayi2 = 5;
int toplam = sayi1 + sayi2;
```

~~KÖTÜ:~~

```
int sayi1 = 5; int sayi2 = 5;
int toplam = sayi1 + sayi2;
```

5.2 "Return" Deyimleri

Fonksiyonlardan değer döndürürken her bir koşul için ayrı ayrı değer döndürmeyin. Koşulları kontrol ettikten sonra fonksiyonunun sonunda tek bir "return" deyimi kullanın.

İYİ:

```
private string MusteriAdiGetir(int musteriNo)
{
      string musteriAd = " ";

      if (musteriNo == 1)
      {
            musteriAd = "Kadir Çamoğlu";
      }
      else
      {
            musteriAd = "Müşteri Bulunamadı";
      }
      return musteriAd;
}
```

~~KÖTÜ:~~

```
private string musteriAdiGetir(int musteriNo)
{
      if (musteriNo == 1)
            return  "Kadir Çamoğlu";
      else
            return "Müşteri Bulunamadı";
}
```

5.3 "If, if-else, if else-if else" Deyimleri

Hemen hemen her programlama dilinde olan deyimlerdir. Bir şart belirtilerek programın ona göre davranmasını sağlar. Bu yüzden bu deyimlerin standartlara göre yazılması oldukça önemlidir.

Öncelikle genel olarak "if" deyimindeki yanlışlıkları ve olması gerekenleri gösterelim.

Genel olarak yapılması gereken “{“ karakterinin şart bitiminden sonra bir satır aşağısında başlatılması “}” karakterinin de olay bitiminden bir satır aşağısında bitirilmelidir.

İYİ:

```
if ( "1" == "1" )
{
      MessageBox.Show( "1 eşittir 1" );
}
```

~~KÖTÜ:~~

```
if ("1"=="1"){
      MessageBox.Show("1 eşittir 1");}
```

Koşul deyiminiz tek satırlık bile olsa blok “{}” içine alın.

~~KÖTÜ:~~

```
if ("1"=="1")
   MessageBox.Show("1 eşittir 1");
```

Karmaşık koşulları “if” bloğu içine yazmayın. Bu tip koşulları blok öncesinde kontrol edilerek sonucun bir değişkenlere yükleyerek “if” koşuluna yazın.

İYİ:

```
enYuksekNotMu = (notum > enYuksekNot);
oncekiEnYuksekNotMu = (notum == enYuksekNot);
gecerliNotmu = (notum < maksimumNot);

if ((enYuksekNotMu && ! oncekiEnYuksekNotMu) &&
gecerliNotMu)
{
      …
}
```

~~KÖTÜ:~~

```
if (((notum > enYuksekNot) && (notum != enYuksekNot))
      && (notum < maximumNot))
{
      …
}
```

Veri tipi bool olan değişkenleri direkt olarak kullanın. “==” ya da “!=” ile tekrar işleme sokmayın.

İYİ:

```
if (donenDeger)
{
}
if (!donenDeger2)
{
}
```

~~KÖTÜ:~~

```
if (donenDeger == true)
{
}

if (donenDeger2 == false)
{
}
```

Koşul yapılarında gereksiz yere koşulun tersini "if" ile kullanmayın. Bunun yerine else kullanın.

İYİ:

```
if (donenDeger)
{
      MessageBox.Show("True Döndü");
}
else
{
      MessageBox.Show("False Döndü");
}
```

~~KÖTÜ:~~

```
if (donenDeger)
{
      MessageBox.Show("True Döndü");
}
if (!donenDeger)
{
      MessageBox.Show("False Döndü");
}
```

İç içe "if" blokları yerine mümkünse "else if" yapısı kullanın.

İYİ:

```
if (ilkDeger > ikinciDeger)
{
      MessageBox.Show("1. değer büyük");
}
else if (ilkDeger < ikinciDeger)
{
      MessageBox.Show("2. değer büyük");
}
else
{
      MessageBox.Show("2 side eşit");
}
```

~~KÖTÜ:~~

```
if (ilkDeger == ikinciDeger)
{
      MessageBox.Show("2 side eşit");
}
else
{
      if (ilkDeger > ikinciDeger)
      {
            MessageBox.Show("1. değer büyük");
      }
      else
      {
            MessageBox.Show("2. değer büyük");
      }
}
```

5.4 "Switch" Deyimleri

Sıralı koşul durumlarının değerlendirilmesine gerek olmayan, belirli bir değere göre belirli bir işlemin yapılması gereken durumlarda "switch" kullanın. Switch deyiminde mümkün olduğunca listeleme (enum) kullanın.

İYİ:

```
public enum CompressionFormat
{
      Zip,
      Gzip,
      Rar
}

public void Compress ( CompressionFormat format )
{
      switch ( format )
      {
            case CompressionFormat.Zip:
                  CompressWithZip();
                  break;
            case CompressionFormat.Gzip:
                  CompressWithGzip();
                  break;
            case CompressionFormat.Rar:
                  CompressWithRar();
                  break;
            default:
                  CompressWithZip();
                  break;
      }
}
```

~~KÖTÜ:~~

```
public enum CompressionFormat
{
	Zip,
	Gzip,
	Rar
}

public void Compress( CompressionFormat format )
{
	if ( format == CompressionFormat.Zip )
	{
		CompressWithZip();
	}
	else if ( format == CompressionFormat.Gzip )
	{
		CompressWithGzip();
	}
	else if ( format == CompressionFormat.Rar )
	{
		CompressWithRar();
	}
	else
	{
		CompressWithZip();
	}
}
```

Mutlaka "default" ile beklenmeyen verileri kontrol edin.

İYİ:

```
public enum CompressionFormat
{
	…
}

public void Compress ( CompressionFormat format )
{
	switch ( format )
	{
		case CompressionFormat.Zip:
			CompressWithZip();
			break;
		…
		default:
			CompressWithZip();
			break;
	}
}
```

~~KÖTÜ:~~

```
public enum CompressionFormat
{
      …
}

public void Compress ( CompressionFormat format )
{
      switch ( format )
      {
            case CompressionFormat.Zip:
                  CompressWithZip();
                  break;
            case CompressionFormat.Gzip:
                  CompressWithGzip();
                  break;
            case CompressionFormat.Rar:
                  CompressWithRar();
                  break;
      }
}
```

5.5 "For / Foreach" Deyimleri

Bu iki döngü tipinde, döngü içinde sadece tek bir satır kod olsa dahi blok parantezlerini kullanın ve döngü değişkenini aksi gerekmedikçe döngü içinde tanımlayın.

İYİ:

```
string[] test = { "bir", "iki", "üç" };
for ( int i = 0; i < test.length; i++ )
{
      MessageBox.Show(test[i]);
}
```

~~KÖTÜ:~~

```
int i;
string[] test = { "bir", "iki", "üç" };
for(i;i<test.length;i++)
      MessageBox.Show(test[i]);
```

İYİ:

```
string test = "bir,iki,uc";
foreach ( string item in test.Split(',') )
{
      MessageBox.Show(item);
}
```

~~KÖTÜ:~~

```
string test = "bir,iki,uc";
string item;
foreach (item in test.split(','))
            MessageBox.Show(item);
```

5.6 "While/do-while" Deyimleri

"while/do-while" döngülerinde, döngünün çıkışını, döngünün koşul deyimiyle kontrol edin. Blok içinde "if" deyimi kullanarak çıkışı zorunlu kalmadıkça kullanmayın.

İYİ:

```
while ( koşul )
{
    // ...
}
…
do
{
    // ...
} while ( koşul );
```

~~KÖTÜ:~~

```
while ( true )
{
    if ( !koşul ) break;
    // ...
}
…
do
{
    // ...
    if ( !koşul ) break;
} while ( true );
```

5.7 İstisnai Durum Yönetimi

Asla boş bir catch bloğu tanımlamayın.
Throw-catch pahalı bir işlemdir. Bunu akış denetimi için kullanmayın.
Değerlendirmeyeceğiniz istisnai durumu yakalamayın.
Catch bloğu içinde try-catch kullanmaktan kaçının.
İstisnai durum yakalanmasını en çok yakalanma olasılığı olandan en azına doğru sıralayın.
Finally bloğunda try için ayrılan kaynağı salıverin.

İYİ:

```
try
{
    // İstisnai durum oluşma ihtimali olan kod
}
catch ( SQLException sx ) // Belirli bir hata
{
    // Oluşan durumu rapor et.
    throw;
}
catch ( Exception ex ) // Diğer hata durumları
{
    throw; // Oluşan durumu rapor et.
}
```

```
    finally
    {
        // try bloğu içinde kullanılan kaynağın salıverilmesi.
    }
```

~~KÖTÜ:~~

```
    try
    {
        // İstisnai durum oluşma ihtimali olan kod
    }
    catch
    {
        throw;
    }
    finally
    {
        // try bloğu içinde kullanılan kaynağın salıverilmesi.
    }
```

İstisnai durumlardan kaçınmak için, öncelikle parametrelerin/verilerin doğruluğunu kontrol etmeyi tercih edin.

İYİ:

```
    if ( baglanti.State != ConnectionState.Closed )
    {
        baglanti.Close();
    }
```

~~KÖTÜ:~~

```
    try
    {
        baglanti.Close();
    }
    catch ( Exception ex )
    {
        // Oluşan durumu rapor et.
    }
```

Catch bloğu içinden farklı bir exception fırlattığınızda mutlaka innerException özelliğine oluşan istisnai durumu ekleyin. Bu şekilde Call Stack zincirini bozmamış olursunuz.

İYİ:

```
try
{
    // İstisnai durum oluşma ihtimali olan kod
}
catch ( Exception ex )
{
    // Oluşan durumu rapor et.
    throw new Exception( "istisnai durum açıklaması",
                                ex );
}
```

~~KÖTÜ:~~

```
try
{
    // İstisnai durum oluşma ihtimali olan kod
}
catch ( Exception ex )
{
    // Oluşan durumu rapor et.
    throw new Exception( "istisnai durum açıklaması" );
}
```

İndeks

Kaynaklar

Bu dokümandaki içerik oluşturulurken aşağıdaki kaynaklardan yararlanılmıştır.

- .Net Framework General Reference: Design Guidelines for Class Library Developers – MSDN Online Reference (http://msdn.microsoft.com/en-us/library/xzf533w0(VS.71).aspx)
- http://se.ethz.ch/teaching/ss2007/251-0290-00/project/CSharpCodingStandards.pdf
- http://www.idesign.net/idesign/download/IDesign%20CSharp%20Coding%20Standard.zip
- http://www.csharpfriends.com/articles/getarticle.aspx?articleid=336
- http://www.dotnetspider.com/tutorials/CodingStandards.doc
- http://msdn.microsoft.com/en-us/library/xzf533w0(VS.71).aspx
- http://10rem.net/articles/net-naming-conventions-and-programming-standards---best-practices

TiKS (Türkçe içerikli Kodlama Standartları) Proje Ekibi

Kadir ÇAMOĞLU

Kadir ÇAMOĞLU, 1974 İstanbul doğumludur. 1995 yılından itibaren profesyonel olarak yazılım projelerinde görev alan ÇAMOĞLU, 1993 yılından sonra eğitim sektöründe de yer almıştır. Microsoft'un yazılım üzerine birçok sertifikasına sahip olan ÇAMOĞLU'nun Pusula, Alfa ve Kodlab yayınevlerinden çıkan on adet kitabı bulunmaktadır. Bilgisayar mühendisliği doktora öğrencisi olan ÇAMOĞLU, BilgeAdam Kurumsal 'da yazılım ve veri tabanı danışmanı olarak çalışmaktadır.

Volkan ATASEVER

Yazılım Dünyasına BASIC ve C/C++ dili üstünde uygulamalar geliştirerek adım atan Volkan daha sonra C# ve VB.NET, Java dilleri ile uygulamalar geliştirmiştir. Haziran 2010 tarihinde ise Bilgisayar Mühendisliği kıyafetini giymiştir. Ineta topluluklarında uzun zaman yazarlık ve editörlük yaptıktan sonra DeveloperMania.Net sitesinin topluluk lideri olmuştur. INETA User Group Lead olarak çeşitli çalışmalar yapmaktadır. Yazılım Uzmanlığı ve Danışmanlığı yapmaktadır. Microsoft'un Akademik sitesi MSAkademik.Net de VB.NET editörlüğü yapmıştır. Konuşmacı olarak çeşitli üniversite ve eğitim merkezinde seminer vermektedir. Microsoft'un Türkiye'de sadece 18 kişiye verdiği Sharepoint Server sertifikasını almıştır. Mobil uygulamalar, ASP.NET, MOSS 2007, SQL Server ve Windows geliştirmeleri içeren projelerde görev almaktadır. Bilgisayar ve teknoloji dergilerinde yazarlık yapmaktadır. Şu anda PCWorld dergisinde yazmaktadır. Ulusal Akademik Konferanslarda akademik bildiri ve makale yazarak bilime katkıda bulunmaktadır. Bilgisayar Mühendisi olarak çalışmalarını sürdürmektedir.

Merve CEYLAN

2001 yılında lisede programlama eğitimi almaya başlayan Merve Ceylan, 2007 yılında Marmara üniversitesi Teknik eğitim fakültesi - bilgisayar teknolojileri ve programlama bölümünü bitirmiştir. Halen eğitimine devam etmekte olup işletme bölümü öğrencisidir.

Üniversiteye devam ederken TÜBİTAK'ta 2 yıl genel muhasebe projesinde yazılım uzmanı olarak çalışmıştır. Şuan 3 yıldır devam etmekte olduğu Maltepe Üniversitesi öğrenci işleri otomasyonu projesinde proje sorumlusu olarak görev almaktadır. Bu yıllar içinde farklı şirketlerin el terminalli restoran otomasyonu ve pazar araştırması projelerinde de çalışmıştır.

Burhan ÇETİNKAYA

2005 senesinde yazılım dünyasına lisede Veri tabanı Programcılığı Bölümüne başlayarak giriş yapmıştır. 2007 senesinde Türkiye çapında gerçekleşen bir yazılım geliştirme yarışmasında Türkiye ikinciliğiyle gümüş madalya kazandı. Birçok orta çapta Windows - Based Client ve Enterprise uygulamalarda proje yöneticisi ve yazılım uzmanı olarak görev aldı. Şuan 1. Sınıf öğrencisi olarak Yeditepe Üniversitesi – Bilgisayar ve Öğretim Teknolojileri Öğretmenliği eğitimime devam etmektedir.

Bahtiyar DİLEK

1984 Amasya doğumlu olan Bahtiyar DİLEK, 16 yaşından beri programlamayla uğraşmaktadır. 2008 yılında Bilimer Bilişim Akademisinden aldığı eğitimle MCPD olmuştur. Sharepoint Developer sertifikası olan Bahtiyar, halen Bilgisayar Mühendisliği öğrencilerine danışmanlık yapmaktadır.

İrfan MEVSİM

1982 Tekirdağ doğumlu olan İrfan Mevsim, Marmara Üniversitesi Bilgisayar ve Kontrol Öğretmenliği 2005 mezunu. 1998 yılından bu yana gelen amatör programlamanın üzerine 2007 yılında Türkiye'nin en büyük beyaz eşya üreticilerinden birinde profesyonel anlamda programlayama başladı. Burada geçen 3 yılın ardından şu anda işletme içinde ihtiyaç duyulan tüm otomasyon ile entegre, mobil ve SAP bağlantılı sistemler üzerine uygulama geliştirebilir durumda.

Can YAPAN

2006 yılında Bulgaristan Varna Teknik Üniversitesinde, Bilgisayar Sistemleri ve Teknolojileri bölümünde eğitimine başlamıştır. Eğitimi sırasında C# dili ve .Net teknolojileri ile ilgilenmiş ve birkaç projede yer almıştır. Halen freelancer yazılım geliştirip eğitimine devam ediyor.

Hakan YAVUZ

2006 senesinde Asp ile Web programcılığına başladım. Ardından Visual Basic 6.0 ile devam edip daha sonrasında tekrar web programlama ile devam etme kararı alıp php ve client tabanlı dilleri öğrenmeye başladım (javascript,ajax) daha sonra .net in gelişmesi ve yaygınlaşması ile beraber c#.net ve vb.net ile devam etmekteyim. Fakat ağırlığım web programlama üstünde.

Yücel YILDIRIM

Yücel YILDIRIM, 1981 Sivas doğumludur.2005 yılı Marmara Üni. Bilgisayar Mühendisliği mezunu olup, beş yılı aşkındır yazılım dünyasında aktif olarak rol almaktadır. Şu an bir bankanın bilgi işlem bölümünde çalışmaktadır.