

12. Hafta İçeriği

- ▶ Genelleyiciler (Generics)
- ▶ Koleksiyonlar (Collections)
 - Collections.Generic Sınıfları
 - Collections.Concurrent Sınıfları
 - Collection Sınıfları
 - Anahtar/Değer Çift Koleksiyonları
 - Bir Koleksiyonu Sıralama

Genelleyiciler (Generics): Temel Kavramlar

- ▶ Genelleyiciler C# 2.0 ile kullanılmaya başlamıştır
- ▶ Genelleyiciler **tip güvenli** veri yapıları oluşturmamıza imkan tanır
- ▶ Tek bir sınıf yazarak tüm tipler için bir genel kalıp oluşturabilirsiniz
- ▶ Kod tekrar kullanımı, tip güvenliği ve performans avantajları getirir
- ▶ Genelleyiciler en çok koleksiyonlarda kullanışlıdır
- ▶ ArrayList sınıfları yerine System.Collections.Generic kullanılması tercih edilmelidir
- ▶ Kendinize ait arayüz, sınıf, yöntem, olay ve temsilciler oluşturabilirsiniz
- ▶ Genelleyicilerle belirli veri tipleri için bazı yöntemlere erişim sınırlandırılabilir

Genelleyiciler Ne Yarar Sağlar-1

- ▶ Genelleyiciler olmadan önce tip dönüştürme ve kutulama (**boxing**) ile çalışma zamanında ortaya çıkan farklı tipteki verilerin işlemleri yapıliyordu
- ▶ Ancak tip dönüştürme (**casting**) ve **boxing** işlemleri büyük performans kaybına neden olmaktadır

```
ArrayList liste1 = new ArrayList(); //Tüm içerik object türünde  
liste1.Add(3); // 3 ve 105 kaydedilirken boxed  
liste1.Add(105);
```

```
ArrayList liste2 = new ArrayList(); //Tüm içerik object türünde  
liste2.Add("Sakarya Üniversitesi"); // String object türüne dönüştürüldü  
liste2.Add("Bilgisayar Mühendisliği");
```

Genelleyiciler Ne Yarar Sağlar-2

- ▶ ArrayList herşeyi **object** türüne dönüştürür ve bu durum aşağıdaki gibi bir hatanın derleyici tarafından bulunmasını imkansız hale getirir. Çalışma zamanında ise program istisna fırlatarak durur.

```
ArrayList liste = new ArrayList();
liste.Add(3);                                //boxing
liste.Add("Sakarya Üniversitesi");           //casting
int t = 0;
foreach (int x in liste)
    t += x;                                  // Bu satır InvalidCastException ile durur
```

Genelleyiciler Ne Yarar Sağlar-3

- ▶ Bize lazım olan şey, çalışma zamanında tipin belirlenebilmesidir ve listelerin tiplerini belirtecek bir parametreye ihtiyaç vardır. Aşağıdaki yaklaşım sorunu çözecektir:

```
List <int> liste1 = new List<int>();           //Artık tipimiz belli, diğerlerine izin yok  
liste1.Add(3);                                //boxing ve casting yok  
liste1.Add("Sakarya Üniversitesi");            //derleyici hatayı yakalar
```

Genelleyiciler Örnek-1

```
public class GenericListe<T> // Burada <T> tipi temsil eder
{
    void Add(T input) { } // T yerine başka bir harf veya kelime de kullanılabilir
} // Çalışma zamanında <T> oluşturulan tip ile yer
class TestGenericList // değiştirilerek geçerli tip ile işlemler yapılır
{
    static void Main()
    {
        GenericListe<int> liste1 = new GenericListe<int>();
        GenericListe<string> liste2 = new GenericListe<string>();
        GenericListe<ExampleClass> liste3 = new GenericListe<ExampleClass>();
    }
}
```

Genelleyicilerde Tip Sınırlama

İstersek genelleyicide kabul edilecek tipleri sınırlayabiliriz.

```
public static void OpTest<T>(T s, T t) where T : class
```

```
{  
    // ...  
}
```

```
public class GenericList<T> where T : struct
```

```
{  
    // ...  
}
```

```
class EmployeeList<T> where T : Employee, IEmployee
```

```
{  
    // ...  
}
```

Genelleyici Yöntemler (Generic Methods)

- Genelleyici yöntem bir tip parametresi ile tanımlanan yöntemdir.
- Genelleyici sınıf için kullanılan tip belirteci ile bu sınıfa ait bir genelleyici yöntemin tip belirleyicisini aynı harf veya kelime seçmeyiniz.
- Tip sınırlandırma yöntemler için de geçerlidir

static void Değiştir <T> (ref T birinci, ref T ikinci)

{

 T temp;

 temp = birinci;

 birinci = ikinci;

 ikinci = temp;

}

public static void TestDeğiştir()

{

 int a = 1;

 int b = 2;

 Değiştir<int>(ref a, ref b);

 Console.WriteLine(a + " " + b);

}

// Değiştir (ref a, ref b); şeklinde de yazılabilir

Koleksiyonlar: Temel Kavamlar

Birbiriyle ilişkili nesneleri tanımlamak, depolamak ve kullanmak için iki yol vardır:

- ◆ Diziler
 - ◆ Koleksiyonlar
- ❑ Diziler, belirli bir sayıdaki nesneler veya temel veri türleri için en iyi seçenekdir.
 - ❑ Koleksiyonlar, bir grup nesne ile çalışırken dizilerden daha esnek bir kullanım sağlarlar.
 - ❑ Dizilerin aksine koleksiyonlar, uygulamanın ihtiyaçlarına göre çalışma zamanında dinamik olarak büyüp küçülebilirler
 - ❑ Ayrıca bazı koleksiyonlarda erişim bir anahtar yardımıyla da kolayca yapılabilir
 - ❑ Koleksiyon nesneniz sadece tek tip elaman içeriyorsa **Collections.Generic** isim uzayındaki sınıfları kullanabilirsiniz. Böylece başka tipte bir verinin koleksiyona eklenmesini önlersiniz. Ayrıca, **Generic** bir koleksiyondan eleman okurken hangi tipte olduğunu sorgulamak zorunda da kalmazsınız.

Basit Koleksiyonlar-1

```
var balıklar=new List<string>();  
balıklar.Add("Hamsi");  
balıklar.Add("Sazan");  
balıklar.Add("Kefal");  
balıklar.Add("Levrek");  
foreach(var balık in balıklar)  
    Console.WriteLine(balık);
```

İstenirse başlangıç değerleri ile bir koleksiyon başlatılabilir

```
var balıklar=new List<string>(){“Hamsi”,“Sazan”,“Kefal”,“Levrek”};  
foreach(var balık in balıklar)  
    Console.WriteLine(balık);
```

Basit Koleksiyonlar-2

```
//Koleksiyondan elemanları çıkartmak  
var sayılar=new List<int>() {0,1,2,3,4,5,6,7,8,9};  
for(int i=sayılar.Count-1; i>=0; i--)  
{  
    if(sayılar[i]%2==1)  
    {  
        sayılar.RemoveAt(i);  
    }  
}  
foreach(int sayı in sayılar)  
{  
    Console.Write(sayı + " ");  
}
```

ÇIKTI:

0 2 4 6 8

Basit Koleksiyonlar-3

```
private static void ListeDöngüsü()
{
    var gezegenler = new List<Gezegen>
    {
        new Gezegen() { Adı="Merkür", Uzaklık=58},
        new Gezegen() { Adı="Venüs", Uzaklık=108},
        new Gezegen() { Adı="Dünya", Uzaklık=150},
        new Gezegen() { Adı="Mars", Uzaklık=228}
    };
    foreach (Gezegen gezegen in gezegenler)
    {
        Console.WriteLine(gezegen.Adı + " " + gezegen.Uzaklık);
    }
}

public class Gezegen
{
    public string Adı{ get; set; }
    public int Uzaklık { get; set; }
}
```

17.04.2017

ÇIKTI:

Merkür 58
Venüs 108
Dünya 150
Mars 228

Basit Koleksiyonlar-4

```
static void Main(string[] args)
{
    List<Vehicle> vehicles = new List<Vehicle>();
    vehicles.Add(new Bicycle());
    vehicles.Add(new Car());
    vehicles.Add(new Truck());

    foreach (Vehicle v in vehicles)
    {
        Console.WriteLine("A {0} has {1} wheels.", v.GetType(), v.Wheels());
    }
}
```

```
public class Car: Vehicle
{
    public override int Wheels()
    {
        return 4;
    }
}

public class Truck : Vehicle
{
    public override int Wheels()
    {
        return 18;
    }
}
```

```
using System;
using System.Collections.Generic;
public abstract class Vehicle
{
    public virtual int Wheels()
    {
        return 0;
    }
}

public class Bicycle : Vehicle
{
    public override int Wheels()
    {
        return 2;
    }
}
```

Collection.Generic İsim Uzayı

Generic İsim Uzayı

Sınıflar:

Comparer<T>, HashSet<T>, LinkedList<T>, SortedSet<T>, Dictionary< TKey, TValue>.ValueCollection

Yapılar

HashSet<T>.Enumerator, Queue<T>.Enumerator, Stack<T>.Enumerator, LinkedList<T>.Enumerator,
KeyValuePair< TKey, TValue>

Arayüzler

ISet<T>, IList<T>, IEnumerable<T>, IReadOnlyList<T>

Collection.Generic Sınıfları

Sınıf İsmi	Açıklama
Dictionary<TKey, TValue>	Anahtar tabanlı organize edilmiş anahtar/değer çiftleri koleksiyonunu temsil eder
List<T>	Bir indeks tarafından erişilen nesne listesini temsil eder. Listelerin üzerinde arama, sıralama ve değiştirme işlemleri için gerekli yöntemleri sağlar.
Queue<T>	FIFO (First-in First-out) tabanlı nesne koleksiyonlarını temsil eder
SortedList<TKey, TValue>	Icomparer<T> gerçeklemesi ile ilişkili anahtarla sıralanmış anahtar/değer çiftleri koleksiyonunu temsil eder
Stack<T>	LIFO (Last-in First-out) tabanlı nesne koleksiyonlarını temsil eder

Collection.Generic Sınıf Örnekleri: List<T>

```
List<int> myInts = new List<int>();  
myInts.Add(1);  
myInts.Add(2);  
myInts.Add(3);  
for (int i = 0; i < myInts.Count; i++)  
{  
    Console.WriteLine("MyInts: {0}", myInts[i]);  
}
```

Collection.Generic Sınıf Örnekleri: Dictionary< TKey, TValue >

```
using System;
using System.Collections.Generic;
public class Customer
{
    public Customer(int id, string name)
    {
        ID = id;
        Name= name;
    }
    public int ID { get; set; }
    public string Name { get ; set; }
}
```

```
class Program{
    static void Main(string[] args){
        List<int> myInts = new List<int>();
        myInts.Add(1);
        myInts.Add(2);
        myInts.Add(3);
        for (int i = 0; i < myInts.Count; i++)
            Console.WriteLine("MyInts: {0}", myInts[i]);
        Dictionary<int, Customer> customers = new Dictionary<int, Customer>();
        Customer cust1 = new Customer(1, "Cust 1");
        Customer cust2 = new Customer(2, "Cust 2");
        Customer cust3 = new Customer(3, "Cust 3");
        customers.Add(cust1.ID, cust1);
        customers.Add(cust2.ID, cust2);
        customers.Add(cust3.ID, cust3);
        foreach (KeyValuePair<int, Customer> custKeyVal in customers)
            Console.WriteLine( "Customer ID: {0}, Name: {1}", custKeyVal.Key, custKeyVal.Value.Name);
    }
}
```

Collection. Concurrent İsim Uzayı

Kullanmış olduğunuz collection sınıflarınız çoklu iş parçacıklarından (multiple thread) erişilecekse Concurrent İsim Uzayı tercih edilmelidir. Ancak Concurrent sınıfları daha kötü performans üretir.

Sınıflar:

BlockingCollection<T>, ConcurrentBag<T>, ConcurrentQueue<T>, ConcurrentStack<T>>,
Partitioner<TSource>, Partitionery, ConcurrentDictionary< TKey, TValue >

Enum

EnumerablePartitionerOptions

Arayüzler

IProducerConsumerCollection<T>

Collection.Concurrent Sınıfları

Sınıf İsmi	Açıklama
BlockingCollection<T>	IProducerConsumerCollection<T> yi gerçekleyen parçacık-güvenli koleksiyonlar için tıkama ve bağlama
ConcurrentBag<T>	Parçacık-güvenli sıralanmamış nesne koleksiyonlarını temsil eder
Partitioner	Ortak bölüntüleme
ConcurrentQueue<T>	Icomparer<T> gerçeklemesi ile ilişkili anahtarla sıralanmış anahtar/değer çiftleri koleksiyonunu temsil eder
ConcurrentStack<T>	LIFO (Last-in First-out) tabanlı nesne koleksiyonlarını temsil eder

Anahtar-Değer Çifti Koleksiyonlarını Gerçekleştirme-1

```
private static void SözlüğeEkle(Dictionary<string, Ele> elm, string sembol, string isim, int atomSayısı)
{
    Ele element = new Ele();
    element.Sembol = sembol;
    element.İsim = isim;
    element.AtomSayısı = atomSayısı;
    elm.Add(key: element.Sembol, value:element);
}
public class Ele
{
    public string Sembol { get; set; }
    public string İsim{ get; set; }
    public int AtomSayısı { get; set; }
}
```

```
public static void Main()
{
    IterateThruDictionary();
}
```

Anahtar-Değer Çifti Koleksiyonlarını Gerçekleştirme-2

```
private static void IterateThruDictionary(){
    Dictionary<string, Ele> elm= BuildDictionary();
    foreach (KeyValuePair<string, Ele> kvp in elm){
        Ele element = kvp.Value;
        Console.WriteLine("Anahtar: " + kvp.Key);
        Console.WriteLine("Değerler: " + element.Sembol + " " + element.İsim + " " + elm.AtomSayısı);
    }
}
private static Dictionary<string, Ele> BuildDictionary(){
    var elm = new Dictionary<string, Ele>();
    SözlüğeEkle(elm, "K", "Potasyum", 19);
    SözlüğeEkle(elm, "Ca", "Kalsiyum", 20);
    SözlüğeEkle(elm, "Sc", "Skandiyum", 21);
    SözlüğeEkle(elm, "Ti", "Titanyum", 22);
    return elm;
}
```

Koleksiyonlarını Sıralama

```
using System;
using System.Collections.Generic;
namespace ArabaSıralama{
    class Program{
        static void Main(string[] args){
            ArabalarıListele();
        }
        private static void ArabalarıListele(){
            var arabalar = new List<Araba>{
                { new Araba() { İsim = "araba1", Renk = "mavi", Hız = 20}},
                { new Araba() { İsim = "araba2", Renk = "sarı", Hız = 50}},
                { new Araba() { İsim = "araba3", Renk = "kırmızı", Hız = 10}},
                { new Araba() { İsim = "araba4", Renk = "yeşil", Hız = 50}},
                { new Araba() { İsim = "araba5", Renk = "beyaz", Hız = 30}},
                { new Araba() { İsim = "araba6", Renk = "beyaz", Hız = 60}},
                { new Araba() { İsim = "araba7", Renk = "siyah", Hız = 50}}
            };
            arabalar.Sort(); //arabaları azalan sırda renk ve hıza göre sırala
            foreach (Araba araba in arabalar){ //Tüm arabaları listele
                Console.Write(araba.Renk.PadRight(5) + " ");
                Console.Write(araba.Hız.ToString() + " ");
                Console.Write(araba.İsim);
                Console.WriteLine();
            }
        }
    }
}
```

```
public class Araba : IComparable<Araba> {
    public string İsim { get; set; }
    public int Hız { get; set; }
    public string Renk { get; set; }
    public int CompareTo(Araba diğer) //IComparable gerçekleme
    {
        int sonuç = String.Compare(this.Renk, diğer.Renk, true);
        if (sonuç == 0) // Eğer renk aynı ise, hızı karşılaştır
        {
            sonuç = this.Hız.CompareTo(düzen.Hız);
            sonuç = -sonuç; // Hız için azalan şekilde sırala
        }
        return sonuç;
    }
}
```

BÖLÜM ALIŞTIRMA ve SORULARI

- ▶ Generic hangi amaçlar için kullanılır?
- ▶ Generic sınıf ve yöntem tanımlamasındaki farklar nelerdir?
- ▶ Generic sınıf ve değerlerin çalışma zamanında nasıl oluşturulduklarını araştırınız
- ▶ Generic tip parametrelerinde neden sınırlandırıcılar kullanılır
- ▶ Genelleyiciler en çok nerelerde kullanışlıdır?
- ▶ Kendinize ait arayüz, sınıf, yöntem, olay ve temsilciler oluşturabilirsiniz
- ▶ Genelleyicilerle belirli veri tipleri için bazı yöntemlere erişim sınırlanabilir
- ▶ Koleksiyonların özellikleri nelerdir
- ▶ Collection.Generic isim uzayı hangi tür sınıflar, arayüzler içerir
- ▶ Collection.Concurrent isim uzayı hangi amaçla kullanılır
- ▶ Collection.Concurrent isim uzayına ait bir sınıfı ve minimum kod kullanarak bir örnek program yazınız