Nesneye Dayalı Programlama

Sakarya Üniversitesi
Bilgisayar ve Bilişim Bilimleri Fakültesi
Bilgisayar Mühendisliği

Prof. Dr. Cemil Öz

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

Genelleyicilerde Tip Sınırlandırma

İstersek genelleyicide kabul edilecek tipleri sınırlandırabiliriz.

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

public class GenericList<T> where T : Employee
{
   // ...
}

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 ilk, ref T ikinci)
{
   T temp;
   temp = birinci;
   birinci = ikinci;
   ikinci = birinci;
}
public static void TestDeğiştir()
{
   int a = 1;
   int b = 2;
   Değiştir<int>(ref a, ref b);          // Değiştir (ref a, ref b); şeklinde de yazılabilir
   Console.WriteLine(a + " " + b);
}
```

Koleksiyonlar: Temel Kavramlar

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 seçenektir.
- Koleksiyonlar, bir grup nesne ile çalışırken dizilerden daha esnek bir kullanım sağlarlar.
- Dizilerin aksine koleksiyonlar, uygulamanın ihtiyaçlarına göre çalışma zamanında dinamik olarak büyüyüp 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.

ArrayList

TheArrayList class supports dynamic arrays, which can grow or shrink as needed.

In C#, standard arrays are of a fixed length, which cannot be changed during program execution. This means you must know in advance how many elements an array will hold.

But sometimes you may not know until runtime precisely how large an array you will need. To handle this situation, use ArrayList.

An ArrayList is a variable-length array of object references that can dynamically increase or decrease in size. An ArrayList is created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array can be shrunk.

ArrayList is currently in wide use in existing code. For this reason, it is examined in depth here. However, many of the same techniques that apply to ArrayList apply to the other collections as well, including the generic collections. ArrayList implements ICollection, IList, IEnumerable, and ICloneable. ArrayList has the constructors

Property	Description
Capacity	Gets or sets the number of elements that the ArrayList can contain.
Count	Gets the number of elements actually contained in the ArrayList.
IsFixedSize	Gets a value indicating whether the ArrayList has a fixed size.
IsReadOnly	Gets a value indicating whether the ArrayList is read-only.
Item	Gets or sets the element at the specified index.

Sr.No.	Methods
1	**public virtual int Add(object value);**Adds an object to the end of the ArrayList.
2	**public virtual void AddRange(ICollection c);**Adds the elements of an ICollection to the end of the ArrayList.
3	**public virtual void Clear();**Removes all elements from the ArrayList.
4	**public virtual bool Contains(object item);**Determines whether an element is in the ArrayList.
5	**public virtual ArrayList GetRange(int index, int count);**Returns an ArrayList which represents a subset of the elements in the source ArrayList.
6	**public virtual int IndexOf(object);**Returns the zero-based index of the first occurrence of a value in the ArrayList or in a portion of it.
7	**public virtual void Insert(int index, object value);**Inserts an element into the ArrayList at the specified index.
8	**public virtual void InsertRange(int index, ICollection c);**Inserts the elements of a collection into the ArrayList at the specified index.
9	**public virtual void Remove(object obj);**Removes the first occurrence of a specific object from the ArrayList.
10	**public virtual void RemoveAt(int index);**Removes the element at the specified index of the ArrayList.
11	**public virtual void RemoveRange(int index, int count);**Removes a range of elements from the ArrayList.
12	**public virtual void Reverse();**Reverses the order of the elements in the ArrayList.
13	**public virtual void SetRange(int index, ICollection c);**Copies the elements of a collection over a range of elements in the ArrayList.
14	**public virtual void Sort();**Sorts the elements in the ArrayList.
15	**public virtual void TrimToSize();**Sets the capacity to the actual number of elements in the ArrayList.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace listarray1
{
    // ArrayList demosu.
    class ArrayListDemo
    {
        static void Main()
        {
            // bir array list oluştur.
            ArrayList al = new ArrayList();
            Console.WriteLine(" ilk eleman sayısı : " + al.Count);
            Console.WriteLine();
            Console.WriteLine(" 6 eleman ekleme  ");
            // array list'e eleman ekleme
            al.Add('C');
            al.Add('A');
            al.Add('E');
            al.Add('B');
            al.Add('D');
            al.Add('F');
            Console.WriteLine(" Eleman sayısı : " + al.Count);
            // dizi indisi kullanılarak dizi elemanlarını görster
            Console.Write(" Su andaki elemanlar ");
            for (int i = 0; i < al.Count; i++)
                Console.Write(al[i] + " ");
            Console.WriteLine("\n");
            Console.WriteLine(" 2 elemanı silme   ");
            // Remove elements from the array list.
            al.Remove('F');
            al.Remove('A');
            Console.WriteLine(" eleman sayısı: " + al.Count);
```

```
            // Use foreach loop to display the list.
            Console.Write(" elemanlar: ");
            foreach (char c in al)
                Console.Write(c + " ");
            Console.WriteLine("\n");
            Console.WriteLine("  20 ilave eleman ekleme  ");
            // Add enough elements to force al to grow.
            for (int i = 0; i < 20; i++)
                al.Add((char)('a' + i));
            Console.WriteLine(" mevcut kapasite: " + al.Capacity);
            Console.WriteLine(" 20 eleman ekledikten sonra eleman sayısı: " + al.Count);
            Console.Write(" elemanlar : ");
            foreach (char c in al)
                Console.Write(c + " ");
            Console.WriteLine("\n");
            // Change contents using array indexing.
            Console.WriteLine(" ilk üç elemanı değiştirme ");
            al[0] = 'X';
            al[1] = 'Y';
            al[2] = 'Z';
            Console.Write(" elemanlar: ");
            foreach (char c in al)
                Console.Write(c + " ");
            Console.WriteLine();
            Console.ReadKey();
        }
    }
}
```

file:///c:/users/cemiloz/documents/visual studio 2015/Projects/listarray1/listarray1/bin/De...

```
ilk eleman sayısı : 0

6 eleman ekleme
Eleman sayısı : 6
Su andaki elemanlar C A E B D F

2 elemanı silme
eleman sayısı: 4
elemanlar: C E B D

 20 ilave eleman ekleme
mevcut kapasite: 32
20 eleman ekledikten sonra eleman sayısı: 24
elemanlar : C E B D a b c d e f g h i j k l m n o p q r s t

ilk üç elemanı değiştirme
elemanlar: X Y Z D a b c d e f g h i j k l m n o p q r s t
```

The SortedList class

represents a collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index.

A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key, it is a Hashtable. The collection of items is always sorted by the key value.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace listarray1
{
    class SortedListDemo
    {
        static void Main()
        {
            SortedList sl = new SortedList();

            sl.Add("001", "Zara Ali");
            sl.Add("002", "Abida Rehman");
            sl.Add("003", "Joe Holzner");
            sl.Add("004", "Mausam Benazir Nur");
            sl.Add("005", "M. Amlan");
            sl.Add("006", "M. Arif");
            sl.Add("007", "Ritesh Saikia");

            if (sl.ContainsValue("Nuha Ali"))
            {
                Console.WriteLine("This student name is already in the list");
            }
            else
            {
                sl.Add("008", "Nuha Ali");
            }

            // get a collection of the keys.
            ICollection key = sl.Keys;

            foreach (string k in key)
            {
                Console.WriteLine(k + ": " + sl[k]);
            }
            Console.ReadKey();
        }
    }
 }
```

file:///c:/users/cemiloz/documents/visual studio 2015/Projects/listarray1/listarray1/bin/De...

```
001: Zara Ali
002: Abida Rehman
003: Joe Holzner
004: Mausam Benazir Nur
005: M. Amlan
006: M. Arif
007: Ritesh Saikia
008: Nuha Ali
```

The Hashtable class

The Hashtable class represents a collection of **key-and-value pairs** that are organized based on the hash code of the key.

It uses the key to access the elements in the collection.

A hash table is used when you need to access elements by using **key**, and you can identify a useful key value. Each item in the hash table has a key/value pair. The key is used to access the items in the collection.

Property	Description
Count	Gets the number of key-and-value pairs contained in the Hashtable.
IsFixedSize	Gets a value indicating whether the Hashtable has a fixed size.
IsReadOnly	Gets a value indicating whether the Hashtable is read-only.
Item	Gets or sets the value associated with the specified key.
Keys	Gets an ICollection containing the keys in the Hashtable.
Values	Gets an ICollection containing the values in the Hashtable.

The following table lists some of the commonly used methods of the Hashtable class:

Sr.No.	Method
1	**public virtual void Add(object key, object value);**Adds an element with the specified key and value into the Hashtable.
2	**public virtual void Clear();**Removes all elements from the Hashtable.
3	**public virtual bool ContainsKey(object key);**Determines whether the Hashtable contains a specific key.
4	**public virtual bool ContainsValue(object value);**Determines whether the Hashtable contains a specific value.
5	**public virtual void Remove(object key);**Removes the element with the specified key from the Hashtable.

```
using System;
using System.Collections;

namespace CollectionsApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      Hashtable ht = new Hashtable();

      ht.Add("001", "Zara Ali");
      ht.Add("002", "Abida Rehman");
      ht.Add("003", "Joe Holzner");
      ht.Add("004", "Mausam Benazir Nur");
      ht.Add("005", "M. Amlan");
      ht.Add("006", "M. Arif");
      ht.Add("007", "Ritesh Saikia");
      if (ht.ContainsValue("Nuha Ali"))
        {
      Console.WriteLine("This student name is already in the list");
        }
        else
        {
          ht.Add("008", "Nuha Ali");
        }

        // Get a collection of the keys.
        ICollection key = ht.Keys;

        foreach (string k in key)
        {
          Console.WriteLine(k + ": " + ht[k]);
        }

        Console.ReadKey();
    }
  }
}
```

Queue.
It represents a first-in, first out collection of object.

It is used when you need a first-in, first-out access of items.

When you add an item in the list, it is called **enqueue**, and when you remove an item, it is called **deque**.

Property	Description
Count	Gets the number of elements contained in the Queue.

methods of the **Queue** class:

Sr.No.	Methods
1	**public virtual void Clear();**Removes all elements from the Queue.
2	**public virtual bool Contains(object obj);**Determines whether an element is in the Queue.
3	**public virtual object Dequeue();**Removes and returns the object at the beginning of the Queue.
4	**public virtual void Enqueue(object obj);**Adds an object to the end of the Queue.
5	**public virtual object[] ToArray();**Copies the Queue to a new array.
6	**public virtual void TrimToSize();**Sets the capacity to the actual number of elements in the Queue.

```
using System;
using System.Collections;

namespace CollectionsApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      Queue q = new Queue();

      q.Enqueue('A');
      q.Enqueue('M');
      q.Enqueue('G');
      q.Enqueue('W');

      Console.WriteLine("Current queue: ");
      foreach (char c in q) Console.Write(c + " ");
      Console.WriteLine();
      q.Enqueue('V');
      q.Enqueue('H');
      Console.WriteLine("Current queue: ");
      foreach (char c in q) Console.Write(c + " ");

      Console.WriteLine();
      Console.WriteLine("Removing some values ");
      char ch = (char)q.Dequeue();
      Console.WriteLine("The removed value: {0}", ch);
      ch = (char)q.Dequeue();
      Console.WriteLine("The removed value: {0}", ch);

      Console.ReadKey();
    }
  }
}
```

currrent queue:
A M G W
Current queue:
A M G W V H
Removing values
The removed value: A
The removed value: M

Stack

It represents a last-in, first out collection of object.

It is used when you need a last-in, first-out access of items.

When you add an item in the list, it is called pushing the item and when you remove it, it is called popping the item.

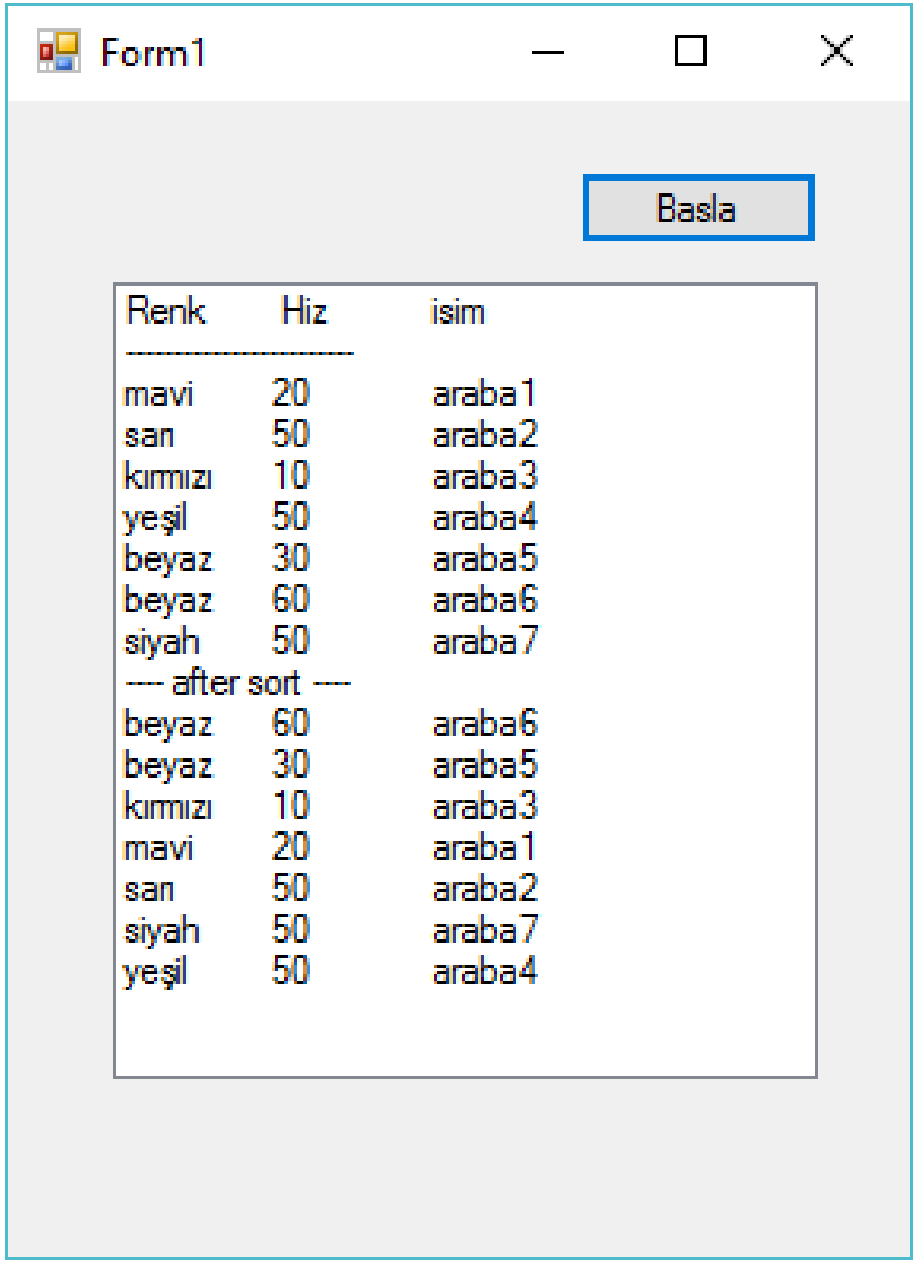
Property	Description
Count	Gets the number of elements contained in the Stack.

methods of the **Stack**class:

Sr.No.	Methods
1	**public virtual void Clear();**Removes all elements from the Stack.
2	**public virtual bool Contains(object obj);**Determines whether an element is in the Stack.
3	**public virtual object Peek();**Returns the object at the top of the Stack without removing it.
4	**public virtual object Pop();**Removes and returns the object at the top of the Stack.
5	**public virtual void Push(object obj);**Inserts an object at the top of the Stack.
6	**public virtual object[] ToArray();**Copies the Stack to a new array.

```
using System;
using System.Collections;

namespace CollectionsApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      Stack st = new Stack();

      st.Push('A');
      st.Push('M');
      st.Push('G');
      st.Push('W');

      Console.WriteLine("Current stack: ");
      foreach (char c in st)
      {
        Console.Write(c + " ");
      }

      Console.WriteLine();

      st.Push('V');
      st.Push('H');

Console.WriteLine("The next poppable value in stack: {0}", st.Peek());
      Console.WriteLine("Current stack: ");
      foreach (char c in st)
      {
        Console.Write(c + " ");
      }

      Console.WriteLine();

      Console.WriteLine("Removing values ");
      st.Pop();
      st.Pop();
      st.Pop();

      Console.WriteLine("Current stack: ");
      foreach (char c in st)
      {
        Console.Write(c + " ");
      }
    }
  }
}
```

```
Current stack:
W G M A
The next poppable value in stack: H
Current stack:
H V W G M A
Removing values
Current stack:
G M A
```

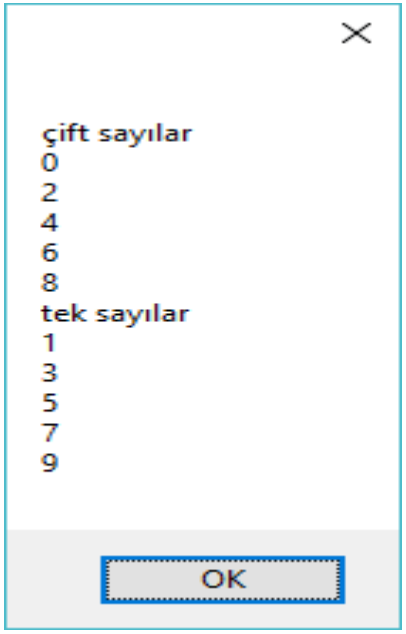

Form1
Basla
Renk Hiz isim
mavi 20 araba1
sarı 50 araba2
kırmızı 10 araba3
yeşil 50 araba4
beyaz 30 araba5
beyaz 60 araba6
siyah 50 araba7
---- after sort ----
beyaz 60 araba6
beyaz 30 araba5
kırmızı 10 araba3
mavi 20 araba1
sarı 50 araba2
siyah 50 araba7
yeşil 50 araba4

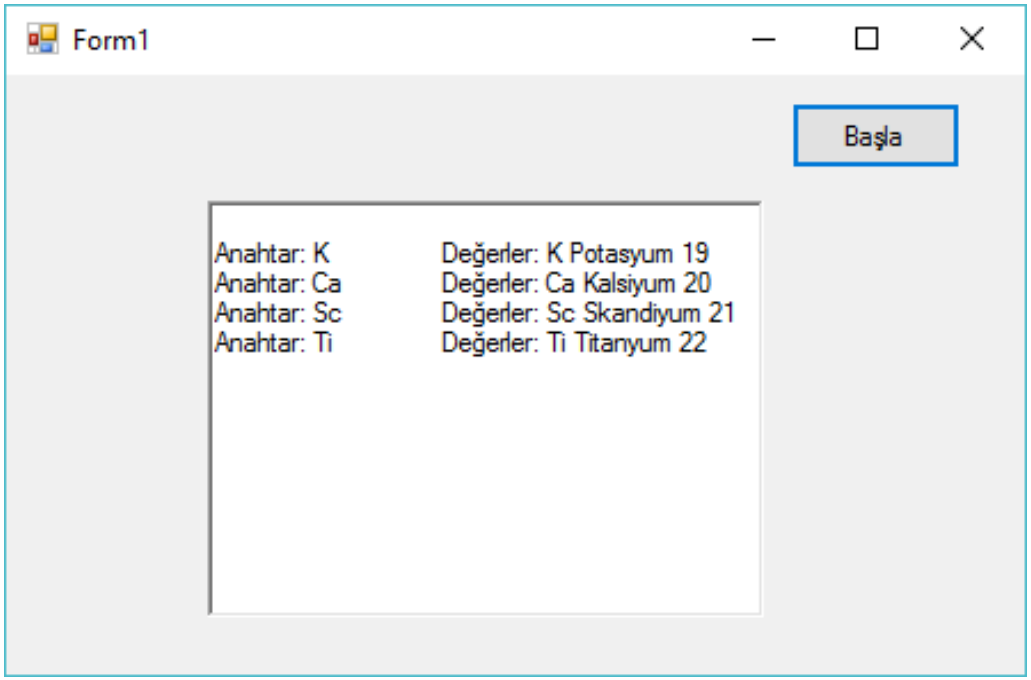
```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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

}
```

```
namespace genelleyiciler
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void ArabalariListele()
        {
            var arabalar = new List<Araba>
            {
                { new Araba() { Isim = "araba1", Renk = "mavi", Hiz = 20}},
                { new Araba() { Isim = "araba2", Renk = "sarı", Hiz = 50}},
                { new Araba() { Isim = "araba3", Renk = "kırmızı", Hiz = 10}},
                { new Araba() { Isim = "araba4", Renk = "yeşil", Hiz = 50}},
                { new Araba() { Isim = "araba5", Renk = "beyaz", Hiz = 30}},
                { new Araba() { Isim = "araba6", Renk = "beyaz", Hiz = 60}},
                { new Araba() { Isim = "araba7", Renk = "siyah", Hiz = 50}}
            };
            listBox1.Items.Add("Renk \t Hiz \t isim ");
            listBox1.Items.Add("------------------------");
            foreach (Araba araba in arabalar)
            {

                listBox1.Items.Add((araba.Renk.PadRight(5)) + "\t" + araba.Hiz.ToString() + "\t " + araba.Isim);

            }
            listBox1.Items.Add("---- after sort ----");
            arabalar.Sort(); //arabaları azalan sırda renk ve hıza göre sırala

            foreach (Araba araba in arabalar)
            {

                listBox1.Items.Add((araba.Renk.PadRight(5)) + "\t" + araba.Hiz.ToString() + "\t " + araba.Isim);

            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            ArabalariListele();
        }
    }
}
```

```
namespace WindowsFormsApplication9
{
    public class Gezegen
    {
        public string Adi { get; set; }
        public int Uzaklik { get; set; }
    }

    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            var gezegenler = new List<Gezegen>
            {
                new Gezegen() { Adi="Merkür", Uzaklik=58},
                new Gezegen() { Adi="Venüs", Uzaklik=108},
                new Gezegen() { Adi="Dünya", Uzaklik=150},
                new Gezegen() { Adi="Mars", Uzaklik=228}
            };
            string msg = " gezegenler \n";
            foreach (Gezegen gezegen in gezegenler)
            {
                msg+= gezegen.Adi + "    " + gezegen.Uzaklik +"\n";
            }
            MessageBox.Show(msg);

        }
    }
}
```

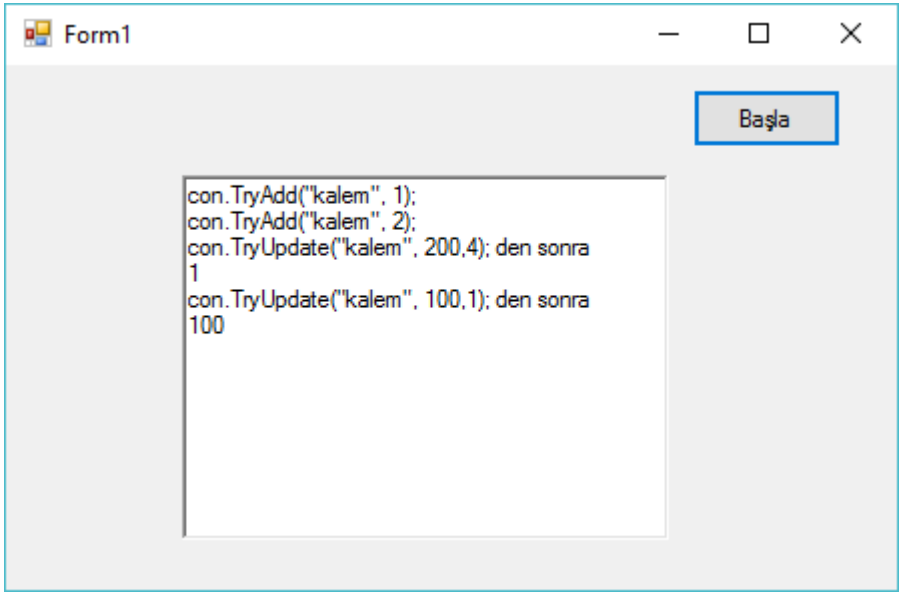
```
namespace WindowsFormsApplication9
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
         private void button1_Click(object sender, EventArgs e)
        {//Koleksiyondan  elemanları çıkartmak
            var sayilar = new List<int>() { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
            var teksayilar = new List<int>();
            for (int i = sayilar.Count - 1; i >= 0; i--)
            {
                if (sayilar[i] % 2 == 1)
                {
                    teksayilar.Add(sayilar[i]);
                    sayilar.RemoveAt(i);
                }
                sayilar.Sort();
                teksayilar.Sort();
            }
            string msg = "çift sayılar\n";
            foreach (int sayi in sayilar)
            {
                msg += sayi.ToString()+"\n";
            }
            msg += "tek sayılar\n";
            foreach (int sayi in teksayilar)
            {
                msg += sayi.ToString() + "\n";
            }
            MessageBox.Show(msg);
        }
    }}
```




Form1
Başla
Anahtar: K
Anahtar: Ca
Anahtar: Sc
Anahtar: Ti
Değerler: K Potasyum 19
Değerler: Ca Kalsiyum 20
Değerler: Sc Skandiyum 21
Değerler: Ti Titanyum 22

```
namespace genelleyiciler1
{
    public partial class Form1 : Form
    {
        public class Ele
        {
            public string Sembol { get; set; }
            public string Isim { get; set; }
            public int AtomSayisi { get; set; }
        }
        public Form1()
        {
            InitializeComponent();
        }

        private void IterateThruDictionary()
        {
            Dictionary<string, Ele> elm = BuildDictionary(); // metod çağrılıyor dönen değer dictionary
            richTextBox1.Text += "\n";
            foreach (KeyValuePair<string, Ele> kvp in elm)
            {
                Ele element = kvp.Value;
                richTextBox1.Text += "Anahtar: " + kvp.Key.ToString() + "\t Değerler: " + element.Sembol + " " + element.Isim + "
" + element.AtomSayisi.ToString()+"\n";
            }
        }
        private Dictionary<string, Ele> BuildDictionary()
        {
            var elm = new Dictionary<string, Ele>();
            SozlugeEkle(elm, "K", "Potasyum", 19);
            SozlugeEkle(elm, "Ca", "Kalsiyum", 20);
            SozlugeEkle(elm, "Sc", "Skandiyum", 21);
            SozlugeEkle(elm, "Ti", "Titanyum", 22);
            return elm;
        }
        private void SozlugeEkle(Dictionary<string, Ele> elm, string sembol, string isim, int atomSayisi)
        {
            Ele element = new Ele();
            element.Sembol = sembol;
            element.Isim = isim;
            element.AtomSayisi = atomSayisi;
            elm.Add(key: element.Sembol, value: element);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            IterateThruDictionary();
        }
    }
}
```

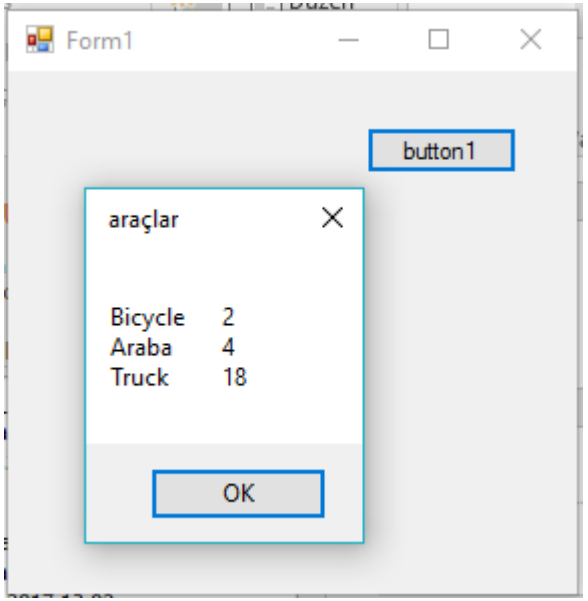
ConcurrentDictionary handles multiple threads. This type from the System.Collections.Concurrent namespace allows multiple threads to access a Dictionary instance. With it, you get a thread-safe, hash-based lookup algorithm.



```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Collections.Concurrent;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace genelleyiciler1
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            var con = new ConcurrentDictionary<string, int>();
            con.TryAdd("kalem", 1);
            richTextBox1.Text += "con.TryAdd(\"kalem\", 1); \n" ;
            con.TryAdd("silgi", 2);
            richTextBox1.Text += "con.TryAdd(\"kalem\", 2); \n";
            // Değer 4 ise güncelle , aşağıdaki ifade isteneni yapmaz).
            con.TryUpdate("kalem", 200, 4);
            richTextBox1.Text += "con.TryUpdate(\"kalem\", 200,4); den sonra \n";
            richTextBox1.Text += con["kalem"]+"\n";
            // Değer 1 ise güncelle , aşağıdaki ifade isteneni yapar).
            con.TryUpdate("kalem", 100, 1);
            richTextBox1.Text += "con.TryUpdate(\"kalem\", 100,1); den sonra \n";
            // Güncel değeri yazdır
            richTextBox1.Text+= con["kalem"];
        }
    }
}
```

```
namespace Genericandpolimorfizm
{
    public partial class Form1 : Form
    {
        public abstract class Vehicle
        {
            public virtual int Wheels()
            {
                return 0;
            }
        }
        public class Bicycle : Vehicle
        {
            public override int Wheels()
            {
                return 2;
            }
        }
        public class Araba : Vehicle
        {
            public override int Wheels()
            {
                return 4;
            }
        }
        public class Truck : Vehicle
        {
            public override int Wheels()
            {
                return 18;
            }
        }
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            List<Vehicle> vehicles = new List<Vehicle>();
            vehicles.Add(new Bicycle());
            vehicles.Add(new Araba());
            vehicles.Add(new Truck());
            string msg = "";
            foreach (Vehicle v in vehicles)
            {
                msg+=v.GetType().Name+"\t"+ v.Wheels().ToString()+"\n";
            }
            MessageBox.Show(msg, " araçlar");

        }
    }
}
```



Generic and operatör overloding and Icomparable struct

```
public interface IComparable<in T>
```

public class Temperature : IComparable<Temperature>

CompareTo(T) method

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object

Örnekte

```
public int CompareTo(Temperature other)
  {
    // If other is not a valid object reference, this instance is greater.
    if (other == null) return 1;

    // The temperature comparison depends on the comparison of
    // the underlying Double values.
    return m_value.CompareTo(other.m_value);
  }
```

```
using System;
using System.Collections.Generic;

public class Temperature : IComparable<Temperature>
{
   // Implement the generic CompareTo method with the Temperature
   // class as the Type parameter.
   //
   public int CompareTo(Temperature other)
   {
      // If other is not a valid object reference, this instance is greater.
      if (other == null) return 1;

      // The temperature comparison depends on the comparison of
      // the underlying Double values.
      return m_value.CompareTo(other.m_value);
   }

   // Define the is greater than operator.
   public static bool operator >  (Temperature operand1, Temperature operand2)
   {
     return operand1.CompareTo(operand2) == 1;
   }
```

```
// Define the is less than operator.
  public static bool operator <  (Temperature operand1, Temperature operand2)
  {
    return operand1.CompareTo(operand2) == -1;
  }

  // Define the is greater than or equal to operator.
  public static bool operator >=  (Temperature operand1, Temperature operand2)
  {
    return operand1.CompareTo(operand2) >= 0;
  }

  // Define the is less than or equal to operator.
  public static bool operator <=  (Temperature operand1, Temperature operand2)
  {
    return operand1.CompareTo(operand2) <= 0;
  }

  // The underlying temperature value.
  protected double m_value = 0.0;

  public double Celsius
  {
     get
     {
        return m_value - 273.15;
     }
  }
```

```
    public double Kelvin
    {
        get
        {
            return m_value;
        }
        set
        {
            if (value < 0.0)
            {
                throw new ArgumentException("Temperature cannot be less than absolute zero.");
            }
            else
            {
                m_value = value;
            }
        }
    }

    public Temperature(double kelvins)
    {
        this.Kelvin = kelvins;
    }
}
```

```
public class Example
{
   public static void Main()
   {
      SortedList<Temperature, string> temps =
         new SortedList<Temperature, string>();

      // Add entries to the sorted list, out of order.
      temps.Add(new Temperature(2017.15), "Boiling point of Lead");
      temps.Add(new Temperature(0), "Absolute zero");
      temps.Add(new Temperature(273.15), "Freezing point of water");
      temps.Add(new Temperature(5100.15), "Boiling point of Carbon");
      temps.Add(new Temperature(373.15), "Boiling point of water");
      temps.Add(new Temperature(600.65), "Melting point of Lead");

      foreach( KeyValuePair<Temperature, string> kvp in temps )
      {
         Console.WriteLine("{0} is {1} degrees Celsius.", kvp.Value, kvp.Key.Celsius);
      }
   }
}
/* This example displays the following output:
     Absolute zero is -273.15 degrees Celsius.
     Freezing point of water is 0 degrees Celsius.
     Boiling point of water is 100 degrees Celsius.
     Melting point of Lead is 327.5 degrees Celsius.
     Boiling point of Lead is 1744 degrees Celsius.
     Boiling point of Carbon is 4827 degrees Celsius.
*/
```


Form1

```
namespace cemildeneme
{
    class mermi
    {
        private int x;
        private int y;
        private int width;

        Image resim;

        public int X
        {
            get
            {
                return x;
            }

            set
            {
                x = value;
            }
        }
        public int Y
        {
            get
            {
                return y;
            }

            set
            {
                y = value;
            }
        }
        public int Width
        {
            get
            {
                return width;
            }

            set
            {
                width = value;
            }
        }

        public mermi()
        {
            resim = Image.FromFile("mermi.png");
            Width = 15;
        }
        public mermi(int _x, int _y)
        {
            X = _x; Y = _y;
            resim = Image.FromFile("mermi.png");
            Width = 15;
        }
        public void mermiCizdir(Graphics g)
        {
            g.DrawImage(resim, X, Y, Width, Width);
        }
    }

}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;

namespace cemildeneme
{
    class savar
    {
        private int x;
        private int y;
        private int width;

        Image resim;

        public int X
        {
            get
            {
                return x;
            }

            set
            {
                x = value;
            }
        }
        public int Y
        {
            get
            {
                return y;
            }

            set
            {
                y = value;
            }
        }
        public int Width
        {
            get
            {
                return width;
            }

            set
            {
                width = value;
            }
        }

        public savar()
        {
            resim = Image.FromFile("savar.png");
            Width = 40;
        }
        public savar(int _x, int _y)
        {
            X = _x; Y = _y;
            resim = Image.FromFile("savar.png");
            Width = 40;
        }
        public void savarCizdir(Graphics g)
        {
            g.DrawImage(resim, X, Y, Width, Width);
        }
    }
}
```

```
namespace cemildeneme
{
    class ucak
    {
        private int x;
        private int y;
        private int width;

        Image resim;

        public int X
        {
            get
            {
                return x;
            }

            set
            {
                x = value;
            }
        }
        public int Y
        {
            get
            {
                return y;
            }

            set
            {
                y = value;
            }
        }
        public int Width
        {
            get
            {
                return width;
            }

            set
            {
                width = value;
            }
        }

        public ucak()
        {
            resim = Image.FromFile("ucak.png");
             Width = 40;
        }
        public ucak(int _x,int _y)
        {
            X = _x;Y = _y;
            resim = Image.FromFile("ucak.png");
            Width = 40;
        }
        public void ucakCizdir(Graphics g)
        {
            g.DrawImage(resim, X, Y, Width, Width);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;



namespace cemildeneme
{
    public partial class Form1 : Form
    {
        List<ucak> ucaklar = new List<ucak>()
        List<mermi> mermiler = new List<mermi
         savar s = new savar(100, 420);
        Graphics g;
        int sayac = 0;
        private Random rnd = new Random();
        public Form1()
        {
            InitializeComponent();
        }
```

```
void cizim()
        {
            g = this.CreateGraphics();
            g.Clear(Color.Black);
            if (ucaklar.Count < 7)
                ucaklar.Add(new ucak(rnd.Next(1, 10) * 40, -40));
            foreach (ucak u in ucaklar)
            {
                //ucak u1 = new ucak();
                u.Y += sayac;
                u.ucakCizdir(g);
                if (u.Y >= 400)
                {
                    timer1.Stop();
                    sayac = 0;
                    MessageBox.Show( "bitti");
                    g.Clear(Color.Black);
                }
            }
            foreach (mermi m in mermiler)
            {
                m.Y -= sayac;
                m.mermiCizdir(g);
            }
            s.savarCizdir(g);
            for(int i=0; i<ucaklar.Count;i++)
            {
                for(int j=0;j< mermiler.Count;j++)
                {
                    if (mermiler[j].X + 15 > ucaklar[i].X && mermiler[j].X < ucaklar[i].X + 40 &&
mermiler[j].Y < ucaklar[i].Y + 40)
                    {
                        ucaklar.RemoveAt(i);
                        mermiler.RemoveAt(j);
                        break;

                    }

                }

            }

        }
        private void Form1_Load(object sender, EventArgs e)
        {

            this.Width = 1000;
            this.Height = 500;
            ucaklar.Add(new ucak(10,10));
            ucaklar.Add(new ucak(30, -30));
            ucaklar.Add(new ucak(50, -50));
```

```
private void Form1_Load(object sender, EventArgs e)
 {

    this.Width = 1000;
    this.Height = 500;
    ucaklar.Add(new ucak(10,10));
    ucaklar.Add(new ucak(30, -30));
    ucaklar.Add(new ucak(50, -50));
    ucaklar.Add(new ucak(100, -10));
    ucaklar.Add(new ucak(150, -50));
    ucaklar.Add(new ucak(100, -100));

    timer1.Interval = 1000;
    timer1.Start();
}

private void Form1_KeyDown(object sender, KeyEventArgs e)
{

    if (e.KeyCode == Keys.Left)
    {
        s.X -= 20;

    }
    if (e.KeyCode == Keys.Right)
    {
        s.X += 20;

    }
    if (e.KeyCode == Keys.Space )
    {
        mermiler.Add(new mermi(s.X+13,s.Y-15));

    }
}
```

```
 private void timer1_Tick(object sender, EventArgs
e)
        {
            this.Enabled = true;
            sayac++;
            if (sayac >= 100)
                timer1.Stop();
            cizim();
        }
    }
}
```