

# Görsel Programlama

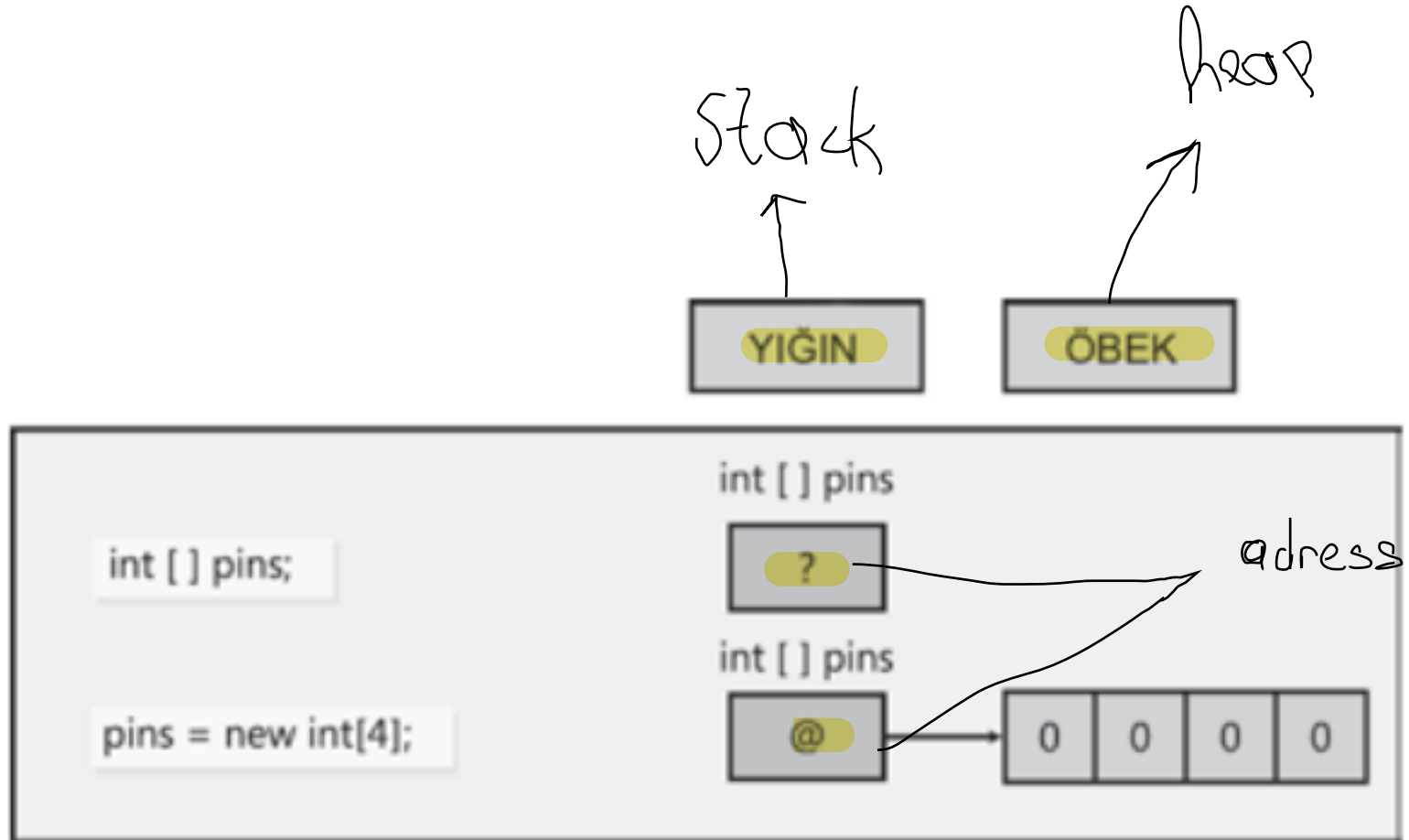
## Ders 3

Dr. Öğr. Üyesi Mehmet Dinçer Erbaş  
Bolu Abant İzzet Baysal Üniversitesi  
Mühendislik Fakültesi  
Bilgisayar Mühendisliği Bölümü

# Diziler ve koleksiyonlar

- Diziler sıralanmış öge serileridir.
- Bir dizideki tüm öğeler aynı türdedir.
- Bir dizinin öğeleri bilgisayarın hafızasında bitişik olarak tutulur ve indeks numaraları kullanılarak her öğeye ulaşılabilir.
- `int[]` numaralar;
  - 📖 Dizinin boyutunu başta bildirmek zorunda değiliz.
- İsteddiğiniz her türde dizi oluşturabilirsiniz.
  - 📖 `Time[]` tarihler;
- Bir diziyi oluşturmak için isim vermek yeterli değildir.
  - 📖 `New` anahtar sözcüğü ile diziyi yaratmanız gerekmektedir.
  - 📖 `numaralar = new int[10];`

# Diziler ve koleksiyonlar





# Diziler ve koleksiyonlar

- Bir diziye `new` anahtar sözcüğü ile oluşturduğunuzda dizinin elemanlarına varsayılan değerler atanır.
  - 📖 Tam sayı ise 0 değeri atanır.
  - 📖 Başka veri tipleri için null, veya false atanır.
- Bir dizi oluşumunun boyutu sabit olmak zorunda değildir.
  - 📖 `int size = int.Parse(Console.ReadLine());`
  - 📖 `int[] numaralar = new int[size];`
- Boyutu 0 olan bir dizi yaratmak mümkündür.
  - 📖 Boyutu dinamik olarak değişen bir diziniz varsa başlangıçta boyut 0 olabilir.
- Dizinin elemanlarına ilk değerler atanabilir.
  - 📖 `int[] numbers = new int[4]{9, 7, 4, 3};`

# Diziler ve koleksiyonlar

- Dizinin elemanlarına bir işlem sonucu olarak hesaplanan ilk değerler atanabilir.

 Random r = new Random();

 int[] numaralar = new int[4]{r.Next() % 10, r.Next() % 10,  
r.Next() % 10, r.Next() % 10};

- Küme parantezi arasındaki değerlerin sayısı, dizinin boyutuna eşit olmalıdır.
- Dizi boyutunu belirtmek zorunlu değildir.

 int[] numaralar = {9, 7, 3, 2};

# Diziler ve koleksiyonlar

- Bir dizi oluştururken veri tipi belirtmek zorunda değilsiniz.

📖 `var isimler = new[]{"Ali", "Veli", "Ayse", "Fatma"};`

- Örnekteki isimler dizisinin string tipinde değişkenler içerdiği derleyici tarafından tespit edilir.

📖 Bu şekilde bir dizi oluşturacaksanız her nesnenin aynı tipte olması gerekir.

- Anonim türlerin dizileri oluşturulabilir.

```
var names = new[] { new { Name = "John", Age = 42 },  
                    new { Name = "Diana", Age = 43 },  
                    new { Name = "James", Age = 15 },  
                    new { Name = "Francesca", Age = 13 } };
```

- Dizi elemanlarına indeks numaraları kullanarak erişebiliriz.
- C dilinde olduğu gibi indeks numaraları 0'dan başlar.
- Dizilerin uzunluğuna `Length` parametresini kullanarak erişebilirsiniz.


# Diziler ve koleksiyonlar

- Dizinin elemanlarına erişim aşağıdaki şekillerde yapılabilir.

```
int[] pins = { 9, 3, 7, 2 };  
for (int index = 0; index < pins.Length; index++)  
{  
    int pin = pins[index];  
    Console.WriteLine(pin);  
}
```

```
int[] pins = { 9, 3, 7, 2 };  
foreach (int pin in pins)  
{  
    Console.WriteLine(pin);  
}
```

- Dizileri amacımız doğrultusunda farklı şekillerde kopyalayabiliriz.

 İki farklı isimle aynı diziye ulaşmak istersek bunu aşağıdaki şekilde yapabiliriz.

```
int[] pins = { 9, 3, 7, 2 };  
int[] alias = pins; // alias ve pins aynı dizi oluşumuna başvurur
```

# Diziler ve koleksiyonlar

- Dizileri amacımız doğrultusunda farklı şekillerde kopyalayabiliriz.

 Elemanları teker teker kopyalayabiliriz.

```
int[] pins = { 9, 3, 7, 2 };  
int[] copy = new int[pins.Length];  
for (int i = 0; i < copy.Length; i++)  
{  
    copy[i] = pins[i];  
}
```

 Bunun dışında kopyalama işlemi için özel fonksiyonlar mevcuttur.

```
int[] pins = { 9, 3, 7, 2 };  
int[] copy = new int[pins.Length];  
pins.CopyTo(copy, 0);
```

```
int[] pins = { 9, 3, 7, 2 };  
int[] copy = (int[])pins.Clone();
```

```
int[] pins = { 9, 3, 7, 2 };  
int[] copy = new int[pins.Length];  
Array.Copy(pins, copy, copy.Length);
```



# Diziler ve koleksiyonlar

- Diziler dışında koleksiyonlar kullanarak öğeleri bir araya getirebiliriz.
  - 📖 Koleksiyonlar `System.Collections` altında oluşturulmuştur.
  - 📖 Koleksiyon sınıfları öğeleri nesne olarak tutar.
    - Yani öğelerin türü object olur.
- ArrayList koleksiyon sınıfı
  - 📖 ArrayList bir dizideki öğelerin sıralarını kaydırma işlemi gerektiğinde verimli çalışan bir sınıftır.
    - Dizilerde bu işlemleri yapmak oldukça zahmetlidir.
  - 📖 ArrayList kullanıldığında, Remove fonksiyonu ile belli bir öğeyi silebilirsiniz.
    - Kaydırma işlemi otomatik olarak yapılır.
  - 📖 Add yöntemi ile yeni bir eleman ekleyebilirsiniz. Boyut yetersiz gelirse otomatik olarak artırılır.

# Diziler ve koleksiyonlar

```
using System;
using System.Collections;
...
ArrayList numbers = new ArrayList();
...
// ArrayList'i doldur
foreach (int number in new int[12]{10, 9, 8, 7, 7, 6, 5, 10, 4, 3, 2, 1})
{
    numbers.Add(number);
}
...
```

# Diziler ve koleksiyonlar

```
// listede sondan bir önceki sıraya bir öğe gir ve sondaki öğeyi yukarı taşı
// (ilk parametre konum;
// ikinci parametre eklenen öğe)
numbers.Insert(numbers.Count-1, 99);

// değeri 7 olan ilk öğeyi kaldır (4. öğe, dizin 3)
numbers.Remove(7);
// şimdi 7. öğe olan öğeyi kaldır, dizin 6 (10)
numbers.RemoveAt(6);

...
// bir for ifadesi kullanarak geri kalan 10 öğede yineleme yap
for (int i = 0; i < numbers.Count; i++)
{
    int number = (int)numbers[i]; // değerın kutulamasını kaldırın dönüştürmeyi gör
    Console.WriteLine(number);
}

...
// iterate remaining 10 using a foreach statement
foreach (int number in numbers) // dönüştürme gerekmez
{
    Console.WriteLine(number);
}
```

# Diziler ve koleksiyonlar

10  
9  
8  
7  
6  
5  
4  
3  
2  
99  
1  
10  
9  
8  
7  
6  
5  
4  
3  
2  
99  
1

# Diziler ve koleksiyonlar

- Queue koleksiyon sınıfı ilk eklenen ilk işleme alınır mantığıyla çalışır
  - 📖 First-in first-out, FIFO.
- Yeni üyeler en sondan sıraya katılır ve sırayı en öndeki terk eder.
  - 📖 Bu amaçla **Enqueue ve Dequeue** fonksiyonları kullanılır.

# Diziler ve koleksiyonlar

```
using System;
using System.Collections;
...
Queue numbers = new Queue();
...
// kuyruğu doldur
foreach (int number in new int[4]{9, 3, 7, 2})
{
    numbers.Enqueue(number);
    Console.WriteLine(number + " has joined the queue");
}
...
// kuyruk boyunca yinele
foreach (int number in numbers)
{
    Console.WriteLine(number);
}
...
// kuyruğu boşalt
while (numbers.Count > 0)
{
    int number = (int)numbers.Dequeue(); // değerin kutulamasını kaldırmak için gerekli dönüştürme
    Console.WriteLine(number + " has left the queue");
}
```

# Diziler ve koleksiyonlar

9 has joined the queue

3 has joined the queue

7 has joined the queue

2 has joined the queue

9

3

7

2

9 has left the queue


3 has left the queue

7 has left the queue

2 has left the queue

# Diziler ve koleksiyonlar

- Stack koleksiyon sınıfı

 Stack koleksiyon sınıfı son eklenen ilk işleme alınır mantığıyla çalışır.

- Last-in First-out (LIFO).

 Yeni üyeler sona eklenir ve son eklenen yığını terk eder.

- Bu amaçla Push ve Pop fonksiyonları kullanılır.



# Diziler ve koleksiyonlar







```
using System;
using System.Collections;
...
Stack numbers = new Stack();
...
// yığını doldur
foreach (int number in new int[4]{9, 3, 7, 2})
{
    numbers.Push(number);
    Console.WriteLine(number + " has been pushed on the stack");
}
...
// yığın boyunca yinele
foreach (int number in numbers)
{
    Console.WriteLine(number);
}
...
// yığını boşalt
while (numbers.Count > 0)
{
    int number = (int)numbers.Pop();
    Console.WriteLine(number + " has been popped off the stack");
}
```

# Diziler ve koleksiyonlar

```
9 has been pushed on the stack
3 has been pushed on the stack
7 has been pushed on the stack
2 has been pushed on the stack
2
7
3
9
2 has been popped off the stack
7 has been popped off the stack
3 has been popped off the stack
9 has been popped off the stack
```

# Diziler ve koleksiyonlar

- Hash tablosu koleksiyon sınıfı

-  Bazı durumlarda tam sayı tipinde değişken ile string, double veya Time türünde değişkenleri bağlamak gerekebilir.
-  Böyle durumlarda hash tabloları kullanmak mümkündür.
-  Bir hash tablosu anahtar/değer çiftleri içerir.
-  Hash tabloları aynı anahtardan iki tane içeremez.
-  Hash tablolarına çok sayıda yeni eleman eklenirse hafızadaki büyüklüğü oldukça fazla artabilir.
-  Hash tablosunda foreach kullanılırsa DictionaryEntry elde edersiniz.
  - Bu kayıttaki key ve value değerlerine bakarak hem anahtara hem de saklanan değere erişebilirsiniz.

# Diziler ve koleksiyonlar

```
using System;
using System.Collections;
...
Hashtable ages = new Hashtable();
...
// Hashtable'ı doldur
ages["John"] = 42;
ages["Diana"] = 43;
ages["James"] = 15;
ages["Francesca"] = 13;
...
// foreach ifadesi boyunca yinele
// Yineleyici key/value çiftini içeren DictionaryEntry nesnesini oluşturur
foreach (DictionaryEntry element in ages)
{
    string name = (string)element.Key;
    int age = (int)element.Value;
    Console.WriteLine("Name: {0}, Age: {1}", name, age);
}
```



# Diziler ve koleksiyonlar

Name: James, Age: 15

Name: John, Age: 42

Name: Francesca, Age: 13

Name: Diana, Age: 43

# Diziler ve koleksiyonlar

- SortedList koleksiyon sınıfı

- 📖 SortedList koleksiyon sınıfı hash tablolarına oldukça benzer.

- Tek farkı nesnelerin anahtarlarına göre sıralanmış olmasıdır.

- 📖 Hash tablosunda olduğu gibi SortedList aynı anahtardan iki tane içeremez.

- 📖 Önceki örneğimizi bu sefer SortedList kullanarak yazalım.

# Diziler ve koleksiyonlar

```
using System;
using System.Collections;
...
SortedList ages = new SortedList();
...
// SortedList'i doldur
ages["John"] = 42;
ages["Diana"] = 43;
ages["James"] = 15;
ages["Francesca"] = 13;
...
// foreach ifadesi boyunca yinele
// yineleyici key/value çiftini içeren DictionaryEntry nesnesini oluşturur
foreach (DictionaryEntry element in ages)
{
    string name = (string)element.Key;
    int age = (int)element.Value;
    Console.WriteLine("Name: {0}, Age: {1}", name, age);
}
```



# Diziler ve koleksiyonlar

Name: Diana, Age: 43

Name: Francesca, Age: 13

Name: James, Age: 15

Name: John, Age: 42



# Dizileri ve koleksiyonlar

- Diziler ve koleksiyonların farkları

- 📖 Dizilerde saklanacak elemanların türleri tanımlanmalıdır. Koleksiyonlarda ise buna gerek yoktur.

- Koleksiyonlar elemanları nesne olarak saklar.

- 📖 Oluşturulan dizinin boyutu sabittir, büyütme veya küçültme mümkün değildir. Koleksiyonlar ise kendilerini dinamik olarak boyutlandırabilir.

- 📖 Bir dizi çok boyutlu olabilir. Ancak koleksiyonlar tek boyutludur.

- Bununla beraber koleksiyonun elemanları başka koleksiyonlar olabilir. Bu yönüyle çok boyutlu dizilere benzer kullanımlar oluşturulabilir.

# Diziler ve koleksiyonlar

- Farklı sayıda girdi ile çalışabilecek fonksiyonlar yazabiliriz.

```
class Util
{
    public static int Min(int[] paramList)
    {
        if (paramList == null || paramList.Length == 0)
        {
            throw new ArgumentException("Util.Min: not enough arguments");
        }
        int currentMin = paramList [0];
        foreach (int i in paramList)
        {
            if (i < currentMin)
            {
                currentMin = i;
            }
        }
        return currentMin;
    }
}
```

# Diziler ve koleksiyonlar

- Farklı sayıda girdi ile çalışabilecek fonksiyonlar yazabiliriz.

```
class Util
{
    public static int Min(params int[] paramList)
    {
        // kod tam olarak önceki gibi
    }
}
```

```
int min = Util.Min(first, second);
```

```
int[] array = new int[2];
array[0] = first;
array[1] = second;
int min = Util.Min(array);
```