1. Overloading (Metot Aşırı Yükleme)

Metot aşırı yükleme, bir sınıfta aynı isme sahip ancak farklı parametreler alan birden fazla metot tanımlama işlemidir. Bu, aynı işlemi farklı türler veya sayıda parametrelerle gerçekleştirmek istediğimizde kullanışlıdır. Aşırı yükleme, derleme zamanında karar verilen bir özelliktir.

Temel Kurallar

- Metotların isimleri aynı olmalı.
- Parametre listesi farklı olmalı (farklı tip, sıra veya sayıda parametre).
- Geri dönüş tipi **tek başına** metotları ayırt etmek için yeterli değildir.

Örnek Sorular

1. **Hesaplama Metodu**: Aynı isimle iki metot yazın, biri iki sayıyı toplasın, diğeri iki metni birleştirsin.

```
using System;

class Calculator
{
    // İki tamsayıyı toplar
    public int Add(int a, int b)
    {
        return a + b;
    }

    // İki ondalık sayıyı toplar
    public double Add(double a, double b)
    {
        return a + b;
    }

    // İki metni birleştirir
    public string Add(string a, string b)
    {
        return a + " " + b;
    }
}

class Program
{
    static void Main()
    {
        Calculator calc = new Calculator();
        Console.WriteLine(calc.Add(5, 10));         // Output: 15
        Console.WriteLine(calc.Add(2.5, 3.7));      // Output: 6.2
        Console.WriteLine(calc.Add("Hello", "World")); // Output: Hello World
    }
}
```

2. Tek Boyutlu ve Çift Boyutlu İndeksleyici Yapılar

İndeksleyiciler, bir nesneye diziymiş gibi erişim sağlar. Özellikle bir sınıf içinde dizi, liste veya matris verilerini kapsüllemek için kullanılır.

Tek Boyutlu İndeksleyici

Bir sınıfa dizi benzeri bir yapı kazandırmak için this anahtar kelimesi kullanılır.

```
class SimpleArray
{
   private int[] data = new int[5];

   public int this[int index]
   {
      get { return data[index]; }
      set { data[index] = value; }
   }
}

class Program
{
   static void Main()
   {
      SimpleArray array = new SimpleArray();
      array[0] = 10;
      array[1] = 20;
      Console.WriteLine(array[0]); // Output: 10
      Console.WriteLine(array[1]); // Output: 20
   }
}
```

Çift Boyutlu İndeksleyici

Bir sınıfı, matris gibi çalıştırmak için çift boyutlu indeksleyici tanımlayabilirsiniz.

```
class Matrix
{
   private int[,] data = new int[3, 3];

   public int this[int row, int col]
   {
      get { return data[row, col]; }
      set { data[row, col] = value; }
   }
}

class Program
{
   static void Main()
   {
      Matrix matrix = new Matrix();
      matrix[0, 0] = 1;
      matrix[1, 1] = 5;
      Console.WriteLine(matrix[0, 0]); // Output: 1
      Console.WriteLine(matrix[1, 1]); // Output: 5
   }
}
```

NOT: İndeksleyiciler, diziler gibi davranır ancak daha fazla kontrol ve özelleştirme imkânı sağlar.

a. Veri Kapsülleme ve Abstraction (Soyutlama)

- İndeksleyiciler, bir nesnenin içindeki verilerin doğrudan erişilmesini engelleyip, bu verilerin kontrollü bir şekilde kullanılmasını sağlar.

Örnek: Doğrudan diziye erişmek yerine bir sınır kontrolü ekleyebilirsiniz.

```
class SecureArray
{
    private int[] data = new int[5];

    public int this[int index]
    {
        get
        {
            if (index < 0 || index >= data.Length)
                throw new IndexOutOfRangeException("Index is out of range.");
            return data[index];
        }
        set
        {
            if (index < 0 || index >= data.Length)
                throw new IndexOutOfRangeException("Index is out of range.");
            data[index] = value;
        }
    }
}

class Program
{
    static void Main()
    {
        SecureArray secureArray = new SecureArray();
        secureArray[0] = 10;
        Console.WriteLine(secureArray[0]); // Output: 10
        // secureArray[5] = 20; // Hata: Index is out of range.
    }
}
```

b. Özelleştirilmiş Mantık Uygulama

- İndeksleyiciler, her erişim sırasında belirli bir mantık çalıştırılmasına olanak tanır. Bu, diziyle mümkün değildir.

Örnek: Verileri çift olarak saklama veya sadece pozitif değerleri kabul etme gibi kısıtlamalar ekleyebilirsiniz.

```
class EvenArray
{
    private int[] data = new int[5];

    public int this[int index]
    {
        get { return data[index]; }
        set
        {
            if (value % 2 != 0)
                throw new ArgumentException("Only even numbers are allowed.");
            data[index] = value;
        }
    }
}
```

```
class Program
{
    static void Main()
    {
        EvenArray evenArray = new EvenArray();
        evenArray[0] = 4; // Geçerli
        Console.WriteLine(evenArray[0]); // Output: 4
        // evenArray[1] = 3; // Hata: Only even numbers are allowed.
    }
}
```

c. Dizi Benzeri Erişim ile Karmaşık Yapılara Ulaşım

- İndeksleyiciler, sadece dizilerde olduğu gibi değil, koleksiyon, sözlük (Dictionary), liste gibi farklı veri yapılarına da dizi benzeri erişim sağlar.

Örnek: Bir kitap koleksiyonundaki kitaplara indeksleyici ile erişim sağlama.

```
class BookCollection
{
    private string[] books = { "C# Guide", "Data Structures", "Algorithms" };

    public string this[int index]
    {
        get
        {
            if (index < 0 || index >= books.Length)
                return "No book found.";
            return books[index];
        }
    }
}

class Program
{
    static void Main()
    {
        BookCollection library = new BookCollection();
        Console.WriteLine(library[1]); // Output: Data Structures
        Console.WriteLine(library[5]); // Output: No book found.
    }
}
```

d. Çift veya Daha Fazla Parametre ile Erişim

- Çift boyutlu veya çok boyutlu veri yapılarına erişim sağlamak için indeksleyiciler daha uygundur.

Örnek: Matris veya tablo gibi bir yapıyı yönetme.

```
class Matrix
{
    private int[,] data = new int[3, 3];

    public int this[int row, int col]
    {
        get { return data[row, col]; }
        set { data[row, col] = value; }
```

```
    }
}

class Program
{
    static void Main()
    {
        Matrix matrix = new Matrix();
        matrix[0, 1] = 5; // İlk satır, ikinci sütuna 5 atanıyor.
        Console.WriteLine(matrix[0, 1]); // Output: 5
    }
}
```

3. Structlar

struct, değer tipi (value type) bir veri yapısıdır. class ile benzer şekilde alanlar, özellikler ve metotlar içerebilir, ancak temel farkları vardır:

- Bellekte **stack** üzerinde saklanır (class ise heap üzerinde).
- Varsayılan olarak struct türleri kopyalanarak geçirilir.
- Miras almayı desteklemez.

Örnek Sorular

1. **Koordinat Yapısı**: X ve Y koordinatlarını temsil eden bir yapı yazın.

```
struct Coordinate
{
    public int X;
    public int Y;

    public Coordinate(int x, int y)
    {
        X = x;
        Y = y;
    }

    public void Display()
    {
        Console.WriteLine($"X: {X}, Y: {Y}");
    }
}

class Program
{
    static void Main()
    {
        Coordinate point = new Coordinate(10, 20);
        point.Display(); // Output: X: 10, Y: 20
    }
}
```

Structlar Ve Classlar Arasındaki Farklar

Struct (yapı) ve class (sınıf) arasındaki fark, genellikle **hafıza yönetimi**, **davranış şekilleri**, ve **kullanım amaçları** açısından ortaya çıkar.

a. Bellek Yönetimi

Struct

- **Değer türüdür** (*Value Type*).
- Bellekte **stack** adı verilen bir alan üzerinde depolanır. Bu, hafif ve hızlı veri türleri için kullanılır.
- Kopyalandığında, yeni bir **tam bağımsız kopya** oluşturur. Yani, bir struct değiştirildiğinde diğer kopyalar bundan etkilenmez.

Örnek:

```
struct Point
{
    public int X;
    public int Y;
}

class Program
{
    static void Main()
    {
        Point p1 = new Point { X = 1, Y = 2 };
        Point p2 = p1; // Kopyalama
        p2.X = 5;

        Console.WriteLine($"p1.X: {p1.X}"); // Output: p1.X: 1
        Console.WriteLine($"p2.X: {p2.X}"); // Output: p2.X: 5
    }
}
```

Class

- **Referans türüdür** (*Reference Type*).
- Bellekte **heap** adı verilen bir alan üzerinde depolanır. Heap, daha büyük ve karmaşık veri türleri için uygundur.
- Kopyalandığında, yalnızca referansı kopyalanır. Yani, iki değişken aynı nesneyi işaret eder ve biri değiştirilirse diğeri de etkilenir.

Örnek:

```
class Point
{
    public int X;
    public int Y;
}

class Program
{
    static void Main()
    {
        Point p1 = new Point { X = 1, Y = 2 };
        Point p2 = p1; // Referansı kopyalama
        p2.X = 5;

        Console.WriteLine($"p1.X: {p1.X}"); // Output: p1.X: 5
        Console.WriteLine($"p2.X: {p2.X}"); // Output: p2.X: 5
```

```
    }
}
```

b. Performans

- **Struct**: Daha hafif ve küçük veriler için uygundur. Örneğin, koordinatlar, zaman damgaları gibi veriler.
- **Class**: Daha karmaşık, çok veri içeren veya sürekli referanslarla çalışılması gereken durumlar için uygundur.

c. Varsayılan Davranışlar

Struct

- Varsayılan olarak **değerleri boş (null) olamaz**. Değer türüdür, dolayısıyla her alanın varsayılan bir değeri olur.
- Bir struct otomatik olarak varsayılan bir kurucu (*constructor*) içerir ve tüm alanlar sıfır değeriyle başlar.

Örnek:

```
struct Point
{
    public int X;
    public int Y;
}

class Program
{
    static void Main()
    {
        Point p = new Point(); // X ve Y otomatik olarak 0 olur.
        Console.WriteLine($"X: {p.X}, Y: {p.Y}"); // Output: X: 0, Y: 0
    }
}
```

Class

- Class içinde alanlar için varsayılan bir değer yoktur. Eğer bir değer atanmazsa, null dönebilir.

d. Inheritance (Kalıtım)

Struct

- **Kalıtım desteklemez**. Bir struct başka bir struct'tan veya sınıftan türemez.
- Sadece **interface**'lerden türeyebilir.

Class

- **Kalıtım destekler**. Sınıflar arasında miras yoluyla özellik ve davranış paylaşımı mümkündür.

e. Kullanım Alanları

Struct:

- Daha basit ve küçük veri yapıları.
- Genellikle **geometrik bilgiler** (koordinatlar, renkler) veya **zaman damgaları** gibi durumlarda kullanılır.

Class:

- Daha karmaşık iş mantığı.
- Özellikle bir nesne yaşam döngüsünü yönetmek, polimorfizm veya kalıtım gibi daha karmaşık yapılar gerektiren durumlar.

4. Enumeration (enum)

enum, bir grup sabit değeri (constants) anlamlı bir şekilde gruplayarak kullanmak için tanımlanır. Örneğin, bir gün değerini "1, 2, 3" yerine anlamlı isimlerle (Monday, Tuesday) ifade edebilirsiniz.

Temel Özellikler

- Varsayılan olarak bir int türüdür.
- Sayısal bir değer atanmamışsa sıfırdan başlar ve sırayla artar.
- İsteğe bağlı olarak farklı bir temel tür atanabilir (byte, short, vb.).

Örnek Sorular

1. **Hava Durumu Enum'u**: Farklı hava durumlarını temsil eden bir enum tanımlayın.

```
enum Weather
{
   Sunny = 1,
   Rainy,
   Cloudy
}

class Program
{
   static void Main()
   {
      Weather today = Weather.Rainy;
      Console.WriteLine($"Today's weather is: {today}"); // Output: Today's weather is: Rainy
      Console.WriteLine($"Weather value: {(int)today}"); // Output: Weather value: 2
   }
}
```

ÖDEVLER

Overloading

1. Matematiksel İşlemleri Çeşitlendiren Bir Fonksiyon

Bir fonksiyon yazın:

- Aynı adla ama farklı parametrelerle toplam işlemi yapacak.
- İlk sürümü iki tam sayıyı toplasın.
- İkinci sürümü üç tam sayıyı toplasın.
- Üçüncü sürümü bir dizi (array) tam sayıyı toplasın.

2. Farklı Şekillerin Alanını Hesaplayan Bir Fonksiyon

Bir fonksiyon yazın:

- Aynı adla ama farklı parametrelerle farklı şekillerin alanını hesaplasın.
- İlk sürüm, bir karenin alanını hesaplasın (bir kenar uzunluğu verilerek).
- İkinci sürüm, bir dikdörtgenin alanını hesaplasın (iki kenar uzunluğu verilerek).
- Üçüncü sürüm, bir dairenin alanını hesaplasın (yarıçap verilerek).

3. Zaman Farkını Farklı Formatlarda Hesaplama

Bir fonksiyon yazın:

- Aynı adla ama farklı parametrelerle iki tarih arasındaki farkı hesaplasın.
- İlk sürüm, gün cinsinden farkı döndürsün.
- İkinci sürüm, saat cinsinden farkı döndürsün.
- Üçüncü sürüm, iki tarih arasında yıl, ay ve gün cinsinden farkı ayrı ayrı döndürsün.

Tek Boyutlu ve Çift Boyutlu İndeksleyiciler

1. Özel Bir Kitaplık Yönetim Sistemi

Bir sınıf oluşturun:

- Sınıf, bir kitap koleksiyonunu (dizi) depolasın.
- Kullanıcı, indeksleyici aracılığıyla belirli bir indekste hangi kitabın olduğunu öğrenebilsin ve kitap adını değiştirebilsin.
- Eğer kullanıcı geçersiz bir indeks ile erişim yaparsa uygun bir hata mesajı döndürsün.

2. Bir Öğrenci Not Sistemi

Bir sınıf oluşturun:

- Öğrencilerin aldığı derslerin notlarını depolasın.
- Kullanıcı indeksleyici aracılığıyla ders adına göre notlara erişebilsin.
- Eğer ders yoksa bir hata mesajı döndürsün.

3. Bir Satranç Tahtası Durumu

Bir sınıf oluşturun:

- Satranç tahtasındaki her bir kareyi temsil etsin.

- Kullanıcı, indeksleyici aracılığıyla bir kareye taş koyabilsin ya da taşın ne olduğunu öğrenebilsin.
- Eğer kullanıcı geçersiz bir kareye erişmeye çalışırsa bir hata mesajı döndürsün.

4. Çok Katmanlı Bir Otopark Sistemi

Bir sınıf oluşturun:

- Her katmanda bir dizi park yerini temsil etsin.
- Kullanıcı, indeksleyici aracılığıyla belirli bir kat ve park yeri kombinasyonuna erişebilsin.
- Park yeri boşsa "Empty" döndürsün, doluysa araç plakasını göstersin.

Struct

1. Zaman İşlemleri

Bir struct oluşturun:

- Saat ve dakikayı tutacak şekilde tasarlansın (Hour ve Minute özellikleri).
- Kullanıcı bir zaman nesnesi oluşturduğunda, eğer girilen saat/dakika geçersizse otomatik olarak 0 yapılmalı.
- Zamanın toplam dakika değerini döndüren bir metot ekleyin.
- İki zaman nesnesi arasındaki farkı (dakika olarak) hesaplayan bir metot ekleyin.

2. Karmaşık Sayı Hesaplama

Bir struct oluşturun:

- Karmaşık sayıları temsil etsin (Real ve Imaginary özellikleri).
- Toplama ve çıkarma işlemlerini gerçekleştiren iki metot ekleyin.
- Sonucu (a + bi) formatında döndüren bir ToString() metodu yazın.

3. GPS Konum Mesafesi

Bir struct oluşturun:

- Enlem ve boylam (Latitude ve Longitude) bilgilerini tutsun.
- İki GPS konumu arasındaki mesafeyi kilometre olarak hesaplayan bir metot ekleyin (Haversine Formülü).

Enum

1. Trafik Işığı Durumu

Bir enum oluşturun:

- Trafik ışıkları (Red, Yellow, Green) için durumları temsil etsin.
- Bir sınıf oluşturun ve enum’a göre hangi durumda ne yapılması gerektiğini döndüren bir metot yazın.

2. Hava Durumu Tahmini

Bir enum oluşturun:

- Hava durumu tiplerini (Sunny, Rainy, Cloudy, Stormy) temsil etsin.
- Enum’a göre tavsiye veren bir metot yazın (örneğin, yağmurluysa şemsiye alın).

3. Çalışan Rolleri ve Maaş Hesaplama

Bir enum oluşturun:

- Çalışan rolleri (Manager, Developer, Designer, Tester) için maaşları belirlemek üzere.
- Enum’a göre maaşı hesaplayan bir metot yazın.