Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **get ve set anahtarları(Properties)**

  Bu anahtar sözcükler kullanılarak sanki iki ayrı metot bildirilmiş gibi olur.

```
class dortgen {
 private int mEn;
public int En{
    get
    {
       return mEn;
    }
    set
    {
    mEn=0;
    }
 }
}
```

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **Operatör Aşırı Yükleme**

  Nesne yönelimli programlamanın en etkin özelliklerinden biri de, sınıflarımız için operatörlere yeni anlamlar yükleyebilmemizdir.

  Örneğin:

  Kompleks tanımlı bir sınıf için, reel ve sanal isimli iki kısımdan oluşan bir sınıfta. İki kompleks sayı üzerinde yapılacak toplama işlemi için operatörleri tekrar yüklenip yeni bir anlam ekleyebiliriz.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **Operatör Aşırı Yükleme**

  Operatör metotları ile ilgili uyulması gerekenler;

  1- Operatör metotları static olarak tanımlanmalıdır.

  2- operator anahtar sözcüğü kullanılır.

  3- Klasik metotlar gibi aşırı yüklenebilir.

  4- ref ve out anahtarları kullanılmamalıdır

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **Aritmetik opertörlerin aşırı yüklenmesi**

```
class Kompleks
{ double mGercek;
  double mSanal;
public double Gercek
{
 get { return mGercek;}
  set { mGercek=value;}
}
```

```
public double Sanal
{
  get { return mSanal;}
  set { mSanal=value;}
}
```

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **Aritmetik opertörlerin aşırı yüklenmesi**

```
public static Kompleks operator+(Kompleks a, Kompleks b)
{
 double gtoplam =a.Gercek + b.Gercek;
 double stoplam= a.Sanal + b.Sanal;
 return new Kompleks(gtoplam,stoplam);
}
```

ile aşırı yüklenmiş olur. Operator ile aşırı yüklenen metot static ve public olmalıdır. Dönüş değeri olarak yeni bir nesne döndürülmüş olur.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **İlişkisel operatörlerin aşırı yüklenmesi**

İlişkisel operatörlerden true ya da false değeri döndürebilir. Bu metotları yazarken dikkat edilmesi gerekir

!=,==,<,>,<=,>= operatörleri ile ilgili tek kural zıt anlamlı operatörlerin her ikisinin de aynı anda yüklenmiş olma zorunluluğudur. Yani operator== bildirilmişse operator!= metodu da bildirilmelidir.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **İlişkisel operatörlerin aşırı yüklenmesi**

```
public static bool operator==(Kompleks a, Kompleks b)
{
 if (a.Sanal==b.Sanal && a.Gercek == b.Gercek)
     return true;
 else
     return false;
}
Kompleks a=new Kompleks(5,8);
Kompleks b=new Kompleks(2,8);
bool c= a==b;
```

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

Kalıtım nesne yönelimli programlama tekniğinin en önemli özelliklerindendir. Kalıtım yoluyla sınıflar birbirlerinden türetilirler. Türeyen sınıflar türediği sınıfın özelliklerini kalıtım yoluyla devraldığı gibi kendisi de yeni özellikler tanımlayabilir.

Örneğin:

Bir hayvan sınıfı oluşturulsun; her hayvanın boy,ağırlık gibi fiziksel özellikleri olsun. Ardından bu sınıftan bir Kedi sınıfı tanımlayalım. Hayvan sınıfında boy ve ağırlığı göstere OzellikGoster() isimli bir metot olsun. Kedi sınıfında da hayvanın kedi olduğunu gösteren string tipinde özel bir değişken tanımlı olsun.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

Temel Sınıflar ve Türetilmiş Sınıflar

Nesne tabanlı programlama ile problemleri gerçek dünyadakine benzer tarzda nesneler şeklinde modelleyerek çözmeye çalışılır.

İnşaat sektöründe bir bina nasıl inşa edilir? Önce projesi çizilir.

Sonra bu proje gerçekleştirilir ve bina ortaya çıkar. Nesne tabanlı programlamada da önce sınıflar (bina örneğindeki projeye benzetilebilir) tasarlanır. Bir uygulama çok sayıda sınıf yardımıyla modellenebilir. Bu sınıflardan bazıları temel sınıf

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

Temel Sınıflar ve Türetilmiş Sınıflar

Bu sınıflardan bazıları temel sınıf bazıları da türetilmiş sınıflardır. Temel sınıf "base" sınıf olarak işlev görür. Türetilmiş sınıf ise temel sınıfın özelliklerine sahip olmanın yanında kendisi temel sınıfta bulunmayan yeni özellikler de geliştirebilir.

Örneğin bir "*okul yönetim uygulaması*" geliştirileceği düşünülsün. İşe okuldaki her şey modellenerek başlanır.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

Temel Sınıflar ve Türetilmiş Sınıflar

Okulumuzda değişik rollerde insanlar vardır. Müdür, müdür yardımcısı, öğretmen, memur, hizmetli, öğrenci rolleri sayılabilir. Her insan rolü için ayrı ayrı sınıf oluşturmak yerine "insan" sınıfı oluşturup sonra bu sınıftan "kalıtım" yoluyla "*müdür*", "*müdür yardımcısı*", "*öğretmen*", "*memur*" ve "*öğrenci*" sınıfları türetilebilir. Bu örnekte "*insan*" sınıfı "*temel sınıf*", diğerleri ise "*türetilmiş sınıf*" olarak adlandırılmaktadır.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

Temel Sınıflar ve Türetilmiş Sınıflar

```
class Insan
{
//temel sınıf üyeleri
}
class Ogrenci : Insan
{
//türetilmiş sınıf üyeleri
}
```

Kalıtım yolu ile public ve protected elemanlar aktarılır. Diğer sınıfların kullanımına kapalı ancak türetme ile türemiş sınıfa geçebilen özellikler protected anahtar sözcüğü kullanılır.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

Temel Sınıflar ve Türetilmiş Sınıflar

Örnekte görüldüğü gibi "Kalıtım yoluyla türetme" : (iki nokta üst üste) işlemci yardımıyla yapılmaktadır. İki noktanın solundaki sınıf (Ogrenci) türetilmiş, sağındaki ise (Insan) temel sınıftır. Burada Ogrenci sınıfı Insan sınıfından türetilmiş bulunmaktadır. Diğer bir ifadeyle Ogrenci sınıfı Insan sınıfının özel (private) olmayan tüm üyelerini kalıtım yoluyla almıştır. Türetme sonucunda Insan sınıfının özel (private) olmayan tüm alanlarını otomatik olarak Ogrenci sınıfı içermektedir.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

Temel Sınıf Kurucularını Çağırma

Türetilmiş bir sınıf, temel sınıfın tüm alanlarına sahiptir. Bu alanların başlangıçta ilk değerlerinin belirlenmesi gerekir. Bu amaçla kurucu metotlar çağrılır. Türetilmiş bir sınıftan temel sınıfa ait kurucu metodu çağırmak için "base" anahtar sözcüğü kullanılır.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

KALITIM(Inheritance)

Temel Sınıf Kurucularını Çağırma

```
class Insan
{
//temel sınıf üyeleri
public Insan(string ad)
{
//işlemler
}
}
```

```
class Ogrenci : Insan
{
//türetilmiş sınıf üyeleri
public Ogrenci(string ad):base(ad)
//Insan(ad) temel sınıf kurucu
metodunu çağırır
{
//...
}
//...
}
```

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

Virtual (Sanal) Metodu

Temel sınıfa ait bir metodun bazen türemiş sınıfta farklı bir şekilde kullanılması gerekebilir. Bu durumda temel sınıfın söz konusu metodu türemiş sınıfın değiştirip kullanabileceği yönünde izin vermesi gerekir. Bu izin, metodun temel sınıfta "virtual" olarak tanımlanması sonucu verilmektedir. Diğer bir ifadeyle temel sınıfta " virtual" olarak tanımlanmış bir metodun türemiş sınıfta yeniden yazılabileceği belirtilmiş olunuyor.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

KALITIM(Inheritance)

Temel Sınıf Kurucularını Çağırma

```
public class Sekil
{
public double PI = 3.14;
protected double x, y;
public Sekil()
{
}
```

```
public Sekil(double x, double y)
{
this.x = x;
this.y = y;
}
public virtual double Alan()
{
return x * y;
}
} //class Sekil sonu
```

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

Virtual (Sanal) Metodu

Sekil sınıfına ait alan adında bir metot olsun. Örnekte alan metodunun "virtual" tanımlandığına dikkat edilmelidir. Sekil sınıfından türeyecek diğer sınıflar içerisinde alan metodunun değiştirilerek yeniden yazılmasına burada izin verilmiş bulunuluyor.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

Override (Geçersiz Kılma) Metodu

Temel sınıfta "virtual" tanımlanmış bir metodun türemiş sınıfta yeniden yazılması için (geçersiz kılma) "override" sözcüğü kullanılır. Temel sınıfta "virtual" tanımlanmamış ya da "static" tanımlanmış olan üyeler (metot, özellik, dizinleyici ya da olay) türemiş sınıfta yeniden yazılamaz (Geçersiz kılınamaz).

KALITIM(Inheritance) Override (Geçersiz Kılma) Metodu

```
public class Sekil
{
public double PI = 3.14;
protected double x, y;
public Sekil()
{
}
public Sekil(double x, double y)
{
this.x = x;
this.y = y;
}
public virtual double Alan()
{
return x * y;
}
} //class Sekil sonu
```

```
public class Daire : Sekil
{
public Daire(double r): base(r, 0)
{
}
public override double Alan()
{
return PI * x * x;
}
}
```

NOT: Ayrıca burada dikkat edilmesi gereken nokta "virtual" ve "override" metotların imzalarının (dönüş türleri ile birlikte parametrelerinin tür ve sayıları) aynı olması gerekliliğidir.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

- **KALITIM(Inheritance)**

  Ayrıca Kedi sınıfında kedinin türünü yazacak bir de metot olsun.

  Türetme aşağıdaki şekilde gerçekleştirilir;

```
class Kedi : Hayvan
{
}
```

Burada Kedi türetilmiş(derived) sınıf, Hayvan da temel (based) sınıftır.

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

KALITIM(Inheritance)

```
class Hayvan //temel sınıf
{
 public double boy;
 public double agirlik;
public void OzellikGoster()
{
 Console.WriteLine(“Boy=“+boy);
 Console.WriteLine(“Agirlik=“+agirlik);
}
}
```

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

KALITIM(Inheritance)

```
class Kedi: Hayvan  //türetilmiş sınıf
{
 public string Turu;
 public void TurGoster()
{
 Console.WriteLine("Tur="+Turu);
 }
}
```

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

KALITIM(Inheritance)

```
class MainMetot
{
 static void Main()
 {
   Kedi k1= new Kedi();
   k1.Agirlik=5;
   k1.Boy=10;
   k1.Turu="van";
   k1.OzellikGoster();
   k1.TurGoster();
} }
```

```
class MainMetot
{
 static void Main()
 {
   Hayvan h1= new Hayvan();
   h1.Agirlik=5;
   h1.Boy=10;
   h1.Turu="van"; //HATA!!!
   h1.OzellikGoster();
   h1.TurGoster(); //HATA!!!
} }
```

Kalıtım ve Operatör Overloading(Aşırı Yükleme)

KALITIM(Inheritance)

Kalıtım yolu ile public ve protected elemanlar aktarılır. Diğer sınıfların kullanımına kapalı ancak türetme ile türemiş sınıfa geçebilen özellikler protected anahtar sözcüğü kullanılır.

Eğer türetme söz konusu değilse protected olarak bilinen elemanlarla private olanlar arasında bir fark olmayacaktır.

SINIF TÜRLERİ

Static Class: Declared with Static keyword, methods in Static Class are also static along with variables of the class.

This class cannot be instantiated, i.e we cannot have objects of this class. To access methods of this class, you can directly use classname.method. Also this class cannot be inherited.

Sealed Class: Declared with Sealed keyword, which enables this class to seal all its variables, methods and properties. No other class can inherit anything from this class or in other words, this class cannot be inherited. But we can instantiate this class, i.e we can have any number of objects of a sealed class.

Abstract Class: Declared with abstract keyword, this class is primarily created as a Inheritable class. An abstract class enables other classes to inherit from this class, but forbids to instantiate. One can inherit from an abstract class but we cannot create objects of an abstract class. Abstract class can have abstract as well as non abstract methods. Abstract methods are those which are not having method definition.

SINIF TÜRLERİ

- **abstract class**

Should be used when there is a IS-A relationship and no instances should be allowed to be created from that abstract class. Example: An Animal is an abstract base class where specific animals can be derived from, i.e. Horse, Pig etc. By making Animal abstract it is not allowed to create an Animal instance.

- **interface**

An interface should be used to implement functionality in a class. Suppose we want a horse to be able to Jump, an interface IJumping can be created. By adding this interface to Horse, all methods in IJumping should be implemented. In IJumping itself only the declarations (e.g. StartJump and EndJump are defined), in Horse the implementations of these two methods should be added.

- **sealed class**

By making Horse sealed, it is not possible to inherit from it, e.g. making classes like Pony or WorkHorse which you like to be inheriting from Horse.

- **static class**

Mostly used for 'utility' functions. Suppose you need some method to calculate the average of some numbers to be used in the Horse class, but you don't want to put that in Horse since it is unrelated and it also is not related to animals, you can create a class to have this kind of methods. You don't need an instance of such a utility class.

- **partial class**

A partial class is nothing more than splitting the file of a class into multiple smaller files. A reason to do this might be to share only part of the source code to others. If the reason is that the file gets too big, think about splitting the class in smaller classes first.