329-Windows Programlama

Bil. Müh. S. Kıvanç EKİCİ

Nesneye Dayalı Programlama

- Yazılımları uygulamalarını tasarlamak için birbiri ile etkileşen nesneleri kullanan paradigma.

- Nesne
  - Bir sınıftan türetilmiş örnek (instance of a class)
  - Durum bilgisi içeren bilgi alanlarına (data fields) sahiptir.
  - Nasıl davranacağını belirleyen metodlar içerir.
  - Diğer nesnelere bilgi gönderebilir ve gelen taleplere göre işlem yapabilir.

Kavramlar

- Sınıf (Class)
- Arayüz (Interface)

- Encapsulation
- Kalıtım (Inheritance)
- Çokbiçimlilik (Polymorphism)

.Net ve Nesneye Dayalı Programlama

- C# ve Class

```
class Car
{
	// The 'state' of the Car.
	public string petName;
	public int currSpeed;
	// The functionality of the Car.
	public void PrintState()
	{
		Console.WriteLine("{0} is going {1} MPH.", petName, currSpeed);
	}
	public void SpeedUp(int delta)
	{
		currSpeed += delta;
	}
}
```

new

```
static void Main(string[] args)
{
    Console.WriteLine("***** Fun with Class
    Types *****\n");
    Car myCar;
    myCar = new Car();
    myCar.petName = "Fred";
}
```

Constructors

- Sınıflardan nesneler bellekte oluşturulmasını ve oluşturulurken bilgi alanlarına ilk değerler atanmasını sağlar.
- Default Constructor

```
class Car
{
// The 'state' of the Car.
public string petName;
public int currSpeed;
// A custom default constructor.
public Car()
{
petName = "Chuck";
currSpeed = 10;
}
```

Constructors-2

```
// Here, currSpeed will receive the
// default value of an int (zero).
public Car(string pn)
{
petName = pn;
}
// Let caller set the full state of the Car.
public Car(string pn, int cs)
{
petName = pn;
currSpeed = cs;
}
...
}
```

this

```
public void SetDriverName(string name)
{
    this.name = name;
}
```

Constructor Chaining

```
class Motorcycle
{
        public int driverIntensity;
        public string driverName;

        // Constructor chaining.
        public Motorcycle() {}

        public Motorcycle(int intensity)
        : this(intensity, "") {}

        public Motorcycle(string name)
        : this(0, name) {}
```

Constructor Chaining -2

```
        // This is the 'master' constructor that does all the real work.
        public Motorcycle(int intensity, string name)
        {
            if (intensity > 10)
            {

                intensity = 10;
            }

            driverIntensity = intensity;
            driverName = name;

        }
        ...
}
```

static

- Bu anahtarla belirlenen alanlar ve metodlar doğrudan sınıf üzerinden erişilirler. Bir nesne örneği oluşturmaya gerek yoktur.
- Böyle bir sınıftan türetilen tüm nesnelerdeki static alanlar aynı değeri alırlar. Yani bir nesnede o alanın değeri değiştirildiğinde tüm diğer nesnelerde de değiştirilmiş olur.
- Console.WriteLine("Much better! Thanks...");

```
class SavingsAccount
{
    // Instance-level data.
    public double currBalance;
    // A static point of data.
    public static double currInterestRate = 0.04;

    public SavingsAccount(double balance)
    {
        currBalance = balance;
    }
}
```

static - 2

```
// Static classes can only
// contain static members!
static class TimeUtilClass
{
        public static void PrintTime()
        { Console.WriteLine(DateTime.Now.ToShortTimeString()); }

        public static void PrintDate()
        { Console.WriteLine(DateTime.Today.ToShortDateString()); }
}
```

Kavramlar - 2

- Encapsulation : Veri tutarlılığını sağlamak ve içsel geliştirim detaylarını gizlemek.
- Kalıtım (inheritance) : Kod yeniden kullanımını desteklemek.
- Çokbiçimlilik (polymorphism) : Benzer türden nesneleri aynı işlemlere tabi tutabilmek.

Encapsulation

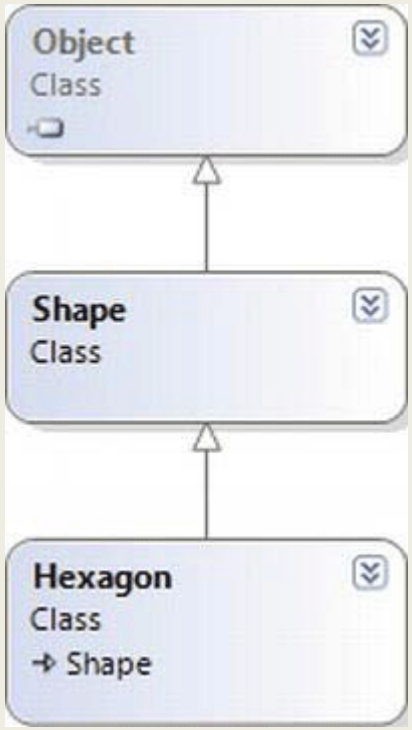
- Nesneyi kullanan açısından gereksiz gerçekleştirim detaylarını gizler.

```
// Assume this class encapsulates the details of opening and closing a database.
DatabaseReader dbReader = new DatabaseReader();
dbReader.Open(@"C:\AutoLot.mdf");
// Do something with data file and close the file.
dbReader.Close();
```

- DatabaseReader sınıfı veritabanı dosyasının bulunması, açılması, yüklenmesi, okunması, değiştirilmesi ve kapatılması ile ilgili işlemleri kullanıcıdan gizler.
- Diğer bir yöntem ise koruma amaçlıdır.
  - private
  - internal
  - protected
  - public

Kalıtım (Inheritance)

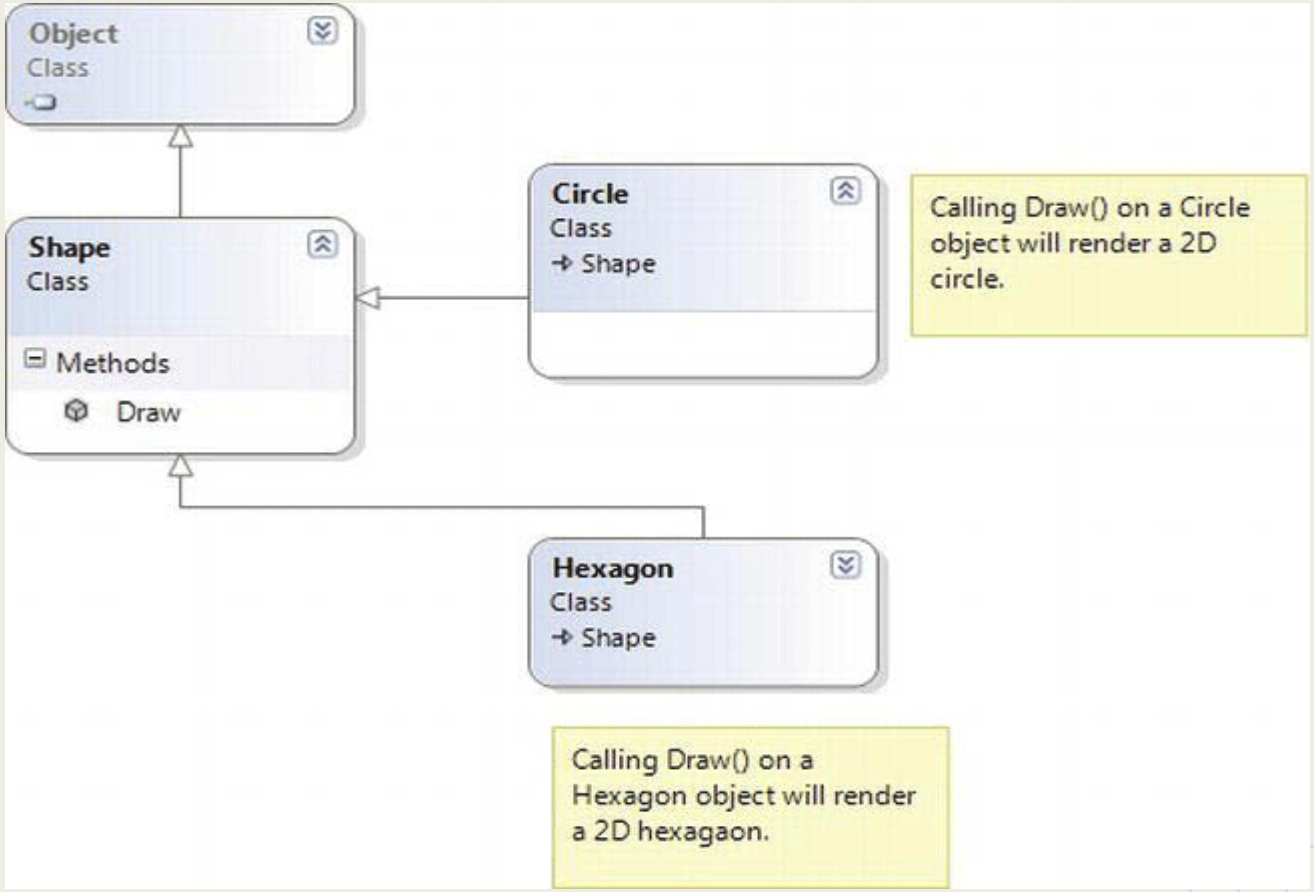
- Halihazırda bulunan sınıfların bilgisini kullanarak yeni sınıflar oluşturma imkanı.



*Pro C# 5.0 and the .NET 4.5 Framework

Çokbiçimlilik (Polymorphism)

- Benzer nesnelere benzer şekilde işlem yapabilme imkanı.



*Pro C# 5.0 and the .NET 4.5 Framework

Erişim Düzenleyiciler (Access Modifiers)

- public
  - Heryerden erişilebilir.
- private
  - Sadece içinde bulunduğu sınıf tarafından erişilebilir.
- protected
  - Sadece içinde bulunduğu sınıf ve bu sınıfın alt sınıflarından erişilebilir.
- internal
  - Sadece içinde bulunduğu assembly (dll,exe..) tarafından erişilebilir.
- protected internal
  - Sadece içinde bulunduğu assembly ve içinde bulunduğu sınıfın alt sınıflarından erişilebilir.

C# Encapsulation

```
class Employee
{
	// Field data.
	private string empName;
	...
	// Accessor (get method).
	public string GetName()
	{
		return empName;
	}
	// Mutator (set method).
	public void SetName(string name)
	{
		// Do a check on incoming value
		// before making assignment.
		if (name.Length > 15)
			Console.WriteLine("Error! Name must be less than 16 characters!");
		else
			empName = name;
	}
}
```

C# Encapsulation - 2

```
class Employee
{
    // Field data.
    private string empName;
    private int empID;
    private float currPay;
    // Properties!
    public string Name
    {
        get { return empName; }
        set
        {
            if (value.Length > 15)
                Console.WriteLine("Error! Name must be less than 16
                characters!");
            else
                empName = value;
        }
    }
```

C# Encapsulation - 3

- ```
  public Employee(string name, int age, int id, float pay)
  {// Humm, this seems like a problem...
      if (name.Length > 15)
          Console.WriteLine("Error! Name must be less than 16 characters!");
      else
          empName = name;
      empID = id;
      empAge = age;
      currPay = pay;
  }
  ```

- ```
  public Employee(string name, int age, int id, float pay)
  {    // Better! Use properties when setting class data.
      // This reduces the amount of duplicate error checks.
      Name = name;
      Age = age;
      ID = id;
      Pay = pay;
  }
  ```

C# Encapsulation - 4

- Readonly

```
public string SocialSecurityNumber
{
        get { return empSS; }
}
```

- Writeonly

```
public string SocialSecurityNumber
{
        set{ empSS=value; }
}
```

C# Encapsulation - 5

- static property

```
// A static point of data.
private static double currInterestRate = 0.04;
// A static property.

public static double InterestRate
{
    get { return currInterestRate; }
    set { currInterestRate = value; }
}
```

C# Encapsulation - 6
Object Initialization Syntax

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

	public Point(int xVal, int yVal)
	{
		X = xVal;
		Y = yVal;
	}

	public Point() { }

	public void DisplayStats()
	{
		Console.WriteLine("[{0}, {1}]", X, Y);
	}
}
```

C# Encapsulation - 7
Object Initialization Syntax -2

```
static void Main(string[] args)
{
        Console.WriteLine("***** Fun with Object Init Syntax *****\n");
        // Make a Point by setting each property manually.
        Point firstPoint = new Point();
        firstPoint.X = 10;
        firstPoint.Y = 10;
        firstPoint.DisplayStats();

        // Or make a Point via a custom constructor.
        Point anotherPoint = new Point(20, 20);
        anotherPoint.DisplayStats();

        // Or make a Point using object init syntax.
        Point finalPoint = new Point { X = 30, Y = 30 };
        finalPoint.DisplayStats();
        Console.ReadLine();
}
```

C# Encapsulation - 8 Initializing Inner Types

```
class Rectangle
{
        private Point topLeft = new Point();
        private Point bottomRight = new Point();
        public Point TopLeft
        {
                get { return topLeft; }
                set { topLeft = value; }
        }
        public Point BottomRight
        {
                get { return bottomRight; }
                set { bottomRight = value; }
        }
        public void DisplayStats()
        {
                Console.WriteLine("[TopLeft: {0}, {1}, {2} BottomRight: {3}, {4}, {5}]",
                topLeft.X, topLeft.Y, topLeft.Color,
                bottomRight.X, bottomRight.Y, bottomRight.Color);
        }
}
```

C# Encapsulation - 8 Initializing Inner Types -2

```
// Create and initialize a Rectangle.
Rectangle myRect = new Rectangle
{
    TopLeft = new Point { X = 10, Y = 10 },
    BottomRight = new Point { X = 200, Y = 200}
};
```

```
// Old-school approach.
Rectangle r = new Rectangle();
Point p1 = new Point();
p1.X = 10;
p1.Y = 10;
r.TopLeft = p1;
Point p2 = new Point();
p2.X = 200;
p2.Y = 200;
r.BottomRight = p2;
```

C# Encapsulation - 9 const

```
namespace ConstData
{
	class MyMathClass
	{
		public const double PI = 3.14;
	}
	class Program
	{
		static void Main(string[] args)
		{
			Console.WriteLine("***** Fun with Const *****\n");
			Console.WriteLine("The value of PI is: {0}", MyMathClass.PI);
			// Error! Can't change a constant!
			// MyMathClass.PI = 3.1444;
			Console.ReadLine();
		}
	}
}
```

C# Encapsulation - 10 readonly

```
class MyMathClass
{
	// Read-only fields can be assigned in ctors,
	// but nowhere else.
	public readonly double PI;
	public MyMathClass ()
	{
		PI = 3.14;
	}
}
```

C# Encapsulation - 11 partial

- Production safhasında bir sınıfın binlerce satır kod içeriği olabilir. Bu kodları çeşitli özelliklere göre gruplayarak farklı dosyalara kayıt edebiliriz.
- Kodlar izole olurlar böylece bir kod grubunu değiştirirken tüm sınıfı kodları etkilenmez.

```
// Employee.cs
class Employee
{
	// Field Data
	// Constructors
	// Methods
	// Properties
}

// Employee2.cs
partial class Employee
{
	// Methods
	// Properties
}
```

C# Kalıtım

```
class Car
{
        public readonly int maxSpeed;
        private int currSpeed;
        public Car(int max)
        {
                maxSpeed = max;
        }
        public Car()
        {
                maxSpeed = 55;
        }
        public int Speed
        {
                get { return currSpeed; }
                set
                {
                currSpeed = value;
                if (currSpeed > maxSpeed)
                {
                        currSpeed = maxSpeed;
                }
        }
}
```

C# Kalıtım - 2
is-a

```
// MiniVan "is-a" Car.
class MiniVan : Car
{
}


static void Main(string[] args)
{
      Console.WriteLine("***** Basic Inheritance *****\n");
      ...
      // Now make a MiniVan object.
      MiniVan myVan = new MiniVan();
      myVan.Speed = 10;
      Console.WriteLine("My van is going {0} MPH",
      myVan.Speed);
      Console.ReadLine();
}
```

C# Kalıtım - 3

- static void Main(string[] args)

```
{
    Console.WriteLine("***** Basic Inheritance *****\n");
    ...
    // Make a MiniVan object.
    MiniVan myVan = new MiniVan();
    myVan.Speed = 10;
    Console.WriteLine("My van is going {0} MPH",
    myVan.Speed);
    // Error! Can't access private members!
    myVan.currSpeed = 55;
    Console.ReadLine();
}

// Illegal! C# does not allow
// multiple inheritance for classes!
```

- class WontWork

```
: BaseClassOne, BaseClassTwo
{}
```

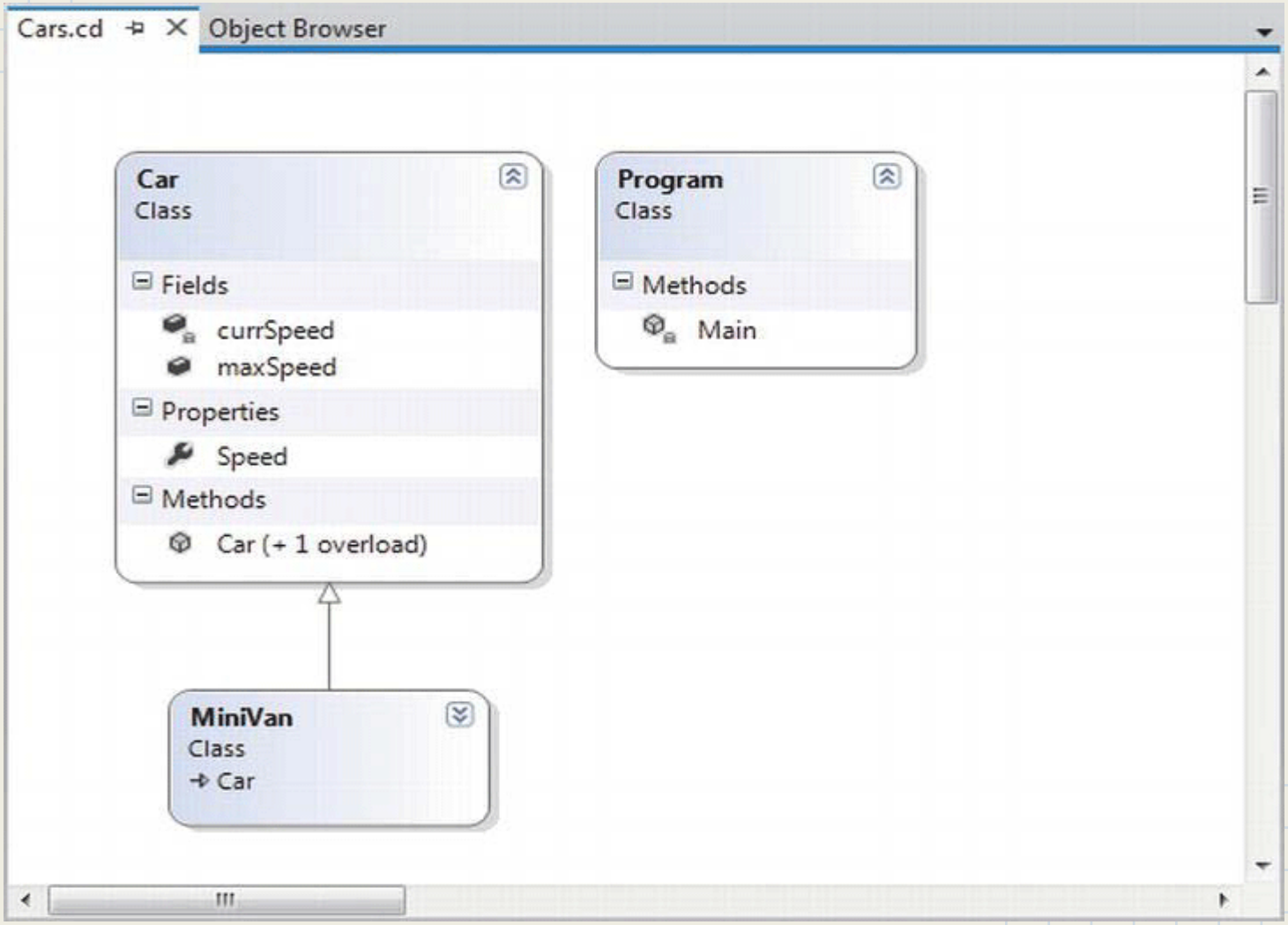
C# Kalıtım - 4
sealed

- sealed ile tanımlanmış olan sınıflar inherit edilemezler.

```
// The MiniVan class cannot be extended!
sealed class MiniVan : Car
{
}


// Error! Cannot extend
// a class marked with the sealed keyword!
class DeluxeMiniVan
: MiniVan
{}
```

Class Diagrams



C# Kalıtım - 5
base

- base kelimesi ile üst sınıfın bir constructor ı çağırılabilir. Böylece üst sınıfta bulunan iş mantığı ve kontroller kullanılmış olur.(Örneğin empID 15 karakterten oluşmak zorunda ise bunu Manager sınıfında tekrardan kontrole gerek yoktur. Üst sınıfta bu kontrol yapılımaktadır.)

```
public Manager(string fullName, int age, int empID,
float currPay, string ssn, int numbOfOpts)
: base(fullName, age, empID, currPay, ssn)
{
    // This property is defined by the Manager class.
    StockOptions = numbOfOpts;
}
```

C# Kalıtım - 6 protected

- Daha önce de bahsedildiği gibi protected elemanlar tüm alt sınıflarda erişilebilmektedir. (private elemanlar alt sınıflarda erişilemez!)

```
partial class Employee
{
// Derived classes can now directly access this information.
protected string empName;
protected int empID;
protected float currPay;
protected int empAge;
protected string empSSN;
...
}
```

C# Kalıtım - 7
has-a

- Kitapta bulunan örnekte temel bir Employee sınıfından , SalesPerson, Manager gibi sınıflar türetilmişti. Bunlar arasında Manager is-a Employee ve SalePerson is-a Employee ilişkisi vardı.
- Bir sınıfın başka bir sınıf türünden bilgi alanı içermesi gerektiğinde is-a ilişkisini kuramayız. Bu durumlarda bir sınıfın diğerini içermesi söz konusudur ve has-a ilişkisi vardır.
- Örnekteki BenefitsPackage sınıfını incelersek Employee elemanlarının bu sınıfı inherit etmesinin mantıksızlığı anlaşılacaktır. Bunun Manager has-a BenefitsPackage ve SalesPerson has-a BenefitsPackage ilişkisi kurulur.

```
// This new type will function as a contained class.
class BenefitPackage
{
    // Assume we have other members that represent
    // dental/health benefits, and so on.
    public double ComputePayDeduction()
    {
    return 125.0;
    }
```

C# Kalıtım - 8
has-a

```
// Employees now have benefits.
partial class Employee
{
	// Contain a BenefitPackage object.
	protected BenefitPackage empBenefits = new BenefitPackage();
	...
}
```

C# Kalıtım - 9
has-a -2

```
public partial class Employee
{
	// Contain a BenefitPackage object.
	protected BenefitPackage empBenefits = new BenefitPackage();

	// Expose certain benefit behaviors of object.
	public double GetBenefitCost()
	{ return empBenefits.ComputePayDeduction(); }

	// Expose object through a custom property.
	public BenefitPackage Benefits
	{
		get { return empBenefits; }
		set { empBenefits = value; }
	}
...
}
```

C# Kalıtım - 10
has-a -3

```
static void Main(string[] args)
{
Console.WriteLine("***** The Employee Class Hierarchy
*****\n");
...
Manager chucky = new Manager("Chucky", 50, 92, 100000,
"333-23-2322", 9000);
double cost = chucky.GetBenefitCost();
Console.ReadLine();
}
```

C# Kalıtım - 11
Nested Types

```
public class OuterClass
{
    // A public nested type can be used by anybody.
    public class PublicInnerClass {}

    // A private nested type can only be used by members
    // of the containing class.
    private class PrivateInnerClass {}
}
```

- Nested sınıflar bir sınıfın tanımı içinde yer alan sınıf tanımlarıdır.
- İçinde bulunduğu sınıfın private alanları dahil bilgilerine erişebilirler.
- private class olarak tanımlanabilir.
- Genellikle yardımcı sınıf olarak oluşturulurlar. Dış dünyada kullanım amacı taşımazlar.

C# Kalıtım - 12
Nested Types -2

```
static void Main(string[] args)
{
        // Create and use the public inner class. OK!
        OuterClass.PublicInnerClass inner;
        inner = new OuterClass.PublicInnerClass();

        // Compiler Error! Cannot access the private class.
        OuterClass.PrivateInnerClass inner2;
        inner2 = new OuterClass.PrivateInnerClass();
}
```

C# Kalıtım - 13
Nested Types -3

```
partial class Employee
{
	public class BenefitPackage
	{
		// Assume we have other members that represent
		// dental/health benefits, and so on.
		public double ComputePayDeduction()
		{
			return 125.0;
		}
	}
	...
}
```

C# Kalıtım - 14
Nested Types -4

```
// Employee nests BenefitPackage.
public partial class Employee
{
    // BenefitPackage nests BenefitPackageLevel.
    public class BenefitPackage
    {
        public enum BenefitPackageLevel
        {
            Standard, Gold, Platinum
        }
        public double ComputePayDeduction()
        {
            return 125.0;
        }
    }
    ...
}
```

- İç içe geçme işlemi gerektiği kadar derin olabilir.

C# Kalıtım - 15
Nested Types -5

```
static void Main(string[] args)
{
        ...
        // Define my benefit level.
        Employee.BenefitPackage.BenefitPackageLevel myBenefitLevel =
        Employee.BenefitPackage.BenefitPackageLevel.Platinum;
        Console.ReadLine();
}
```

C# Polymorphism

```
public partial class Employee
{
	public void GiveBonus(float amount)
	{
		Pay += amount;
	}
...
}
```

C# Polymorphism - 2

```
static void Main(string[] args)
{
        Console.WriteLine("***** The Employee Class Hierarchy *****\n");
        // Give each employee a bonus?
        Manager chucky = new Manager("Chucky", 50, 92, 100000, "333-23-2322", 9000);
        chucky.GiveBonus(300);
        chucky.DisplayStats();
        Console.WriteLine();

        SalesPerson fran = new SalesPerson("Fran", 43, 93, 3000, "932-32-3232", 31);
        fran.GiveBonus(200);
        fran.DisplayStats();
        Console.ReadLine();
}
```

- Problem : GiveBonus herkes için eşit şekilde uygulanıyor. Oysaki bir Manager veya SalesPerson için farklı olmalı örneğin satış sayıları dikkate alınmalıydı.

C# Polymorphism - 3 virtual - override

- virtual ile tanımlanan metodlar temel bir işlem sunarlar ancak kalıtım ile oluşturulan alt sınıflarda aynı isimde eklenerek bu methodun içeriği değiştirilebilir anlamına gelir.
- Eğer alt sınıfta bu metod yeniden eklenmemiş ise tanımlanan temel işlem uygulanır.
- Yeniden yazma işlemi için "override" kelimesi kullanılır.
-

```
partial class Employee
{
    // This method can now be "overridden" by a derived class.
    public virtual void GiveBonus(float amount)
    {
        Pay += amount;
    }
    ...
}
```

C# Polymorphism - 4 virtual - override -2

```
class SalesPerson : Employee
{
        ...
        // A salesperson's bonus is influenced by the number of sales.
        public override void GiveBonus(float amount)
        {
                int salesBonus = 0;
                if (SalesNumber >= 0 && SalesNumber <= 100)
                        salesBonus = 10;
                else
                {
                        if (SalesNumber >= 101 && SalesNumber <= 200)
                                salesBonus = 15;
                        else
                                salesBonus = 20;
                }
                base.GiveBonus(amount * salesBonus);
        }
}
```

C# Polymorphism - 5 virtual - override - 3

```
class Manager : Employee
{
        ...
        public override void GiveBonus(float amount)
        {
                base.GiveBonus(amount);
                Random r = new Random();
                StockOptions += r.Next(500);
        }
}
```

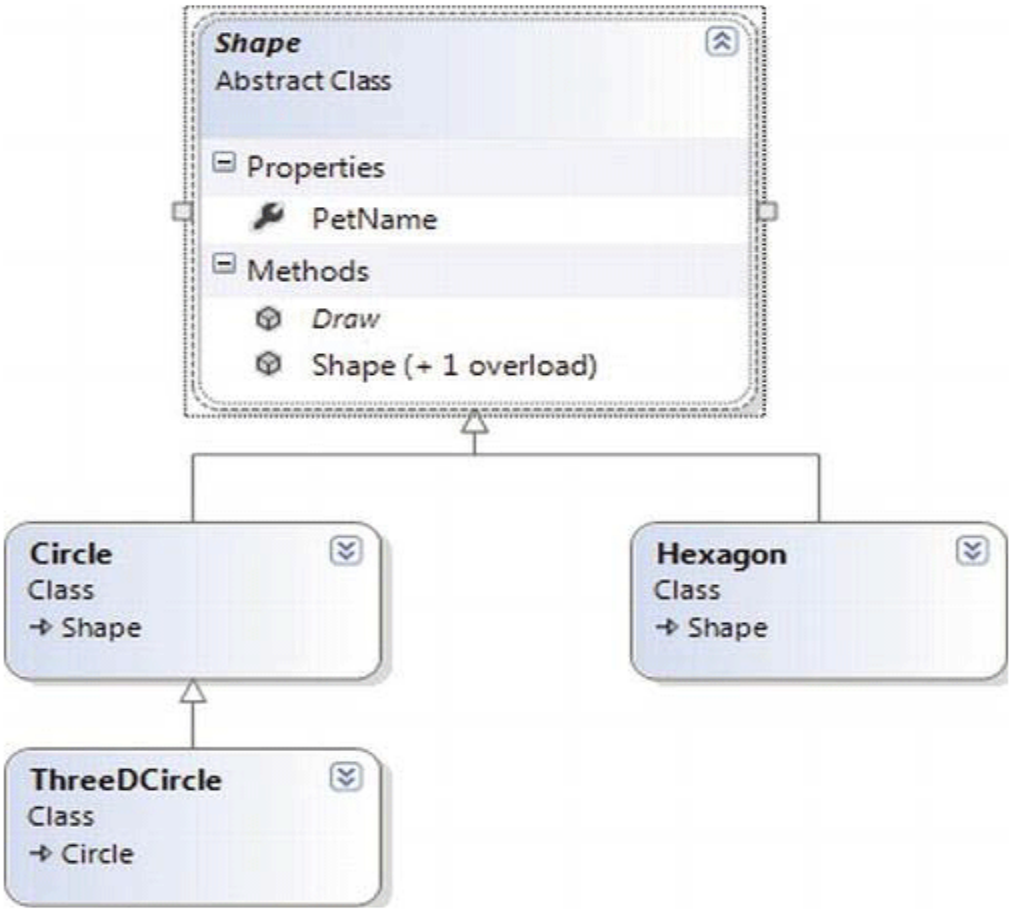
C# Polymorphism - 6 Abstract Class

- Abstract sınıflar sadece kalıtım amacıyla oluşturulmuş sınıflardır. Bu sınıflardan nesne türetilmez.

```
// What exactly does this mean?
Employee X = new Employee();


// Error! Cannot create an instance of an abstract class!
Employee X = new Employee();
```

C# Polymorphism - 7
Abstract Class - 2



*Pro C# 5.0 and the .NET 4.5 Framework

C# Polymorphism - 8
Abstract Class - 3

```
// The abstract base class of the hierarchy.
abstract class Shape
{
	public Shape(string name = "NoName")
	{ PetName = name; }

	public string PetName { get; set; }
	// A single virtual method.
	public virtual void Draw()
	{
		Console.WriteLine("Inside Shape.Draw()");
	}
}
```

C# Polymorphism - 9
Abstract Class - 4

```
// Circle DOES NOT override Draw().
class Circle : Shape
{
	public Circle() {}
	public Circle(string name) : base(name){}
}

// Hexagon DOES override Draw().
class Hexagon : Shape
{
	public Hexagon() {}
	public Hexagon(string name) : base(name){}
	public override void Draw()
	{
	Console.WriteLine("Drawing {0} the Hexagon", PetName);
	}
}
```

C# Polymorphism - 10
Abstract Class - 5

```
abstract class Shape
{
    // Force all child classes to define how to be rendered.
    public abstract void Draw();
    ...
}
```

C# Polymorphism - 11
Abstract Class - 6

```
static void Main(string[] args)
{
	// Make an array of Shape-compatible objects.
	Shape[] myShapes = {new Hexagon(), new Circle(), new Hexagon("Mick"),
	new Circle("Beth"), new Hexagon("Linda")};

	// Loop over each item and interact with the
	// polymorphic interface.

	foreach (Shape s in myShapes)
	{
		s.Draw();
	}

	Console.ReadLine();
}
```

C# Polymorphism - 12 Casting

- Casting bir nesneyi başka bir sınıf türüne dönüştürme işlemidir.
- Casting sadece kalıtımsal olarak allta veya üstte bulunan sınıflar arasında yapılabilir.

```
object frank = new Manager("Frank Zappa", 9, 3000, 40000, "111-11-1111", 5);

// A Manager "is-an" Employee too.
Employee moonUnit = new Manager("MoonUnit Zappa", 2, 3001, 20000, "101-11-1321", 1);

// A PTSalesPerson "is-a" SalesPerson.
SalesPerson jill = new PTSalesPerson("Jill", 834, 3002, 100000, "111-12-1119", 90);

GivePromotion((Manager)frank);

SalesPerson jane = (SalesPerson) someEmployee;
```

C# Polymorphism - 13 Casting - as - 2

```
// Catch a possible invalid cast.
try
{
	Hexagon hex = (Hexagon)frank;
}
catch (InvalidCastException ex)
{
	Console.WriteLine(ex.Message);
}


// Use "as" to test compatability.
Hexagon hex2 = frank as Hexagon;

if (hex2 == null)
	Console.WriteLine("Sorry, frank is not a Hexagon...");
```

C# Polymorphism - 14 Casting - is - 3

```
static void GivePromotion(Employee emp)
{
    Console.WriteLine("{0} was promoted!", emp.Name);
    if (emp is SalesPerson)
    {
        Console.WriteLine("{0} made {1} sale(s)!", emp.Name,
        ((SalesPerson)emp).SalesNumber);

    }

    if (emp is Manager)
    {
        Console.WriteLine("{0} had {1} stock options...", emp.Name,
        ((Manager)emp).StockOptions);

    }
}
```

System.Object

- Bütün sınıfların varsayılan olarak inherit edildiği temel sınıf. Oluşturulan tüm sınıflar burada bulunan eleman ve methodları içerir.

```
public class Object
{
    // Virtual members.
    public virtual bool Equals(object obj);
    protected virtual void Finalize();
    public virtual int GetHashCode();
    public virtual string ToString();
    // Instance-level, nonvirtual members.
    public Type GetType();
    protected object MemberwiseClone();
    // Static members.
    public static bool Equals(object objA, object objB);
    public static bool ReferenceEquals(object objA, object objB);
}
```

Ödev

- Code Project
  - Chapter 4: Core C# Programming Constructs, Part II bölümünde bulunan kod örnekleri bilgisayarda çalıştırılacak. Ortaya çıkan projeleri içeren solution klasörü Moodle dan yüklenecek.
- Haftalık Rapor
  - Chapter 5: Understanding Encapsulation kısmı özetlenecek.

SON