Nesneye Dayalı Programlama Deney Föyü

Sınıf nedir

Nesneler için ortak şablon niteliğindedir. Kendisini kullanarak oluşturulacak nesnelerin ortak özellikleri ve davranışları ortaya koymak için gereklidir.

```
namespace Ornek1
{
    class Ogrenci
    {
        /* Sınıf içerisinde field, property, metod ve olaylar tanımlanabilir.*/
        public int numara { get; }
        public sinif { get; }
        public Ogrenci(int no, int sNo)
        {
            numara = no;
            sinif = sNo;
        }
    }
}
```

Ödev 1:

a. *Field, property, get-set, method, event, constructor kavramlarını kendi cümleleriniz ile detaylı şekilde açıklayınız. Bu kavramlar nedir, ne zaman kullanılır.*

b. *Verilen boyutlardaki dikdörtgeni çizdiren uygulamayı yazınız. (En uzunluğu 10-100 piksel arasında ve boy uzunluğu da, en uzunluğunun, iki katı kadar olmalıdır.) Bu kavramları kodunuz içerisinde kullanınız ve kod üzerinde açıklama satırı ile her birini gösteriniz.*

Nesne nedir

Belli özellikleri ve davranışı olan herşey nesne olarak adlandırılabilir. Örneğin kedi, hayvanlar sınıfının bir nesnesi olabilir. Diğer bir deyişle özelliklerin tanımlandığı sınıfa göre gerçeklenen yapılardır.

```
using System;

namespace Ornek1
{
    class Program
    {
        static void Main(string[] args)
        {
            Ogrenci ogrenciAli = new Ogrenci(1648, 2);
            Console.WriteLine("Ali adli ogrencinin numarasi\t:" + ogrenciAli.numara
+ "\nAli adli ogrencinin sinifi\t:" + ogrenciAli.sinif);
            Console.Read();
        }
    }
}
```

Encapsulation

- Data ile kodu aynı yerden yönetir böylece veriye dışarıdan erişim olmaz.
- Erişmek isteyenler ise veriye erişim izni olan nesne aracılığı ile erişebilir.
- Amacı verinin dışarıdan gizlenmesi ve erişiminin kısıtlanmasıdır.
- Güvenli ve güvenilir kod için gereklidir.

```
class CPUkullan
    {
        int CPUYuzde;
        public CPUkullan() { CPUYuzde = 100; }
        public int cpuYuzde {

            get { return CPUYuzde; }

            set
            {
                if (value > 0 && value < 20 && CPUYuzde>value)
                    CPUYuzde = CPUYuzde - value;
                else
                    Console.WriteLine("\t %"+value + "\'lik islemci kullanim
istegi fazla oldugundan, kabul edilmedi.");
            }

        }
    }
```

***Soru**: Encapsulation işlemi get-set haricinde nasıl yapılabilir?*

***Ödev 2.** Encapsulation faydalarını örnek kodlar ile açıklayınız.*

Polymorphism

Genel anlamı ile bir adın birbiriyle alakalı fakat teknik açıdan farklı iki veya daha fazla amaç için kullanılabilmesi yeteneğidir.

Örneğin C dilinde, mutlak değer bulma işlemi için üç farklı fonksiyon tanımlıdır: abs(), labs() ve fabs(). Fakat çok biçimliliği destekleyen C++'da bu fonksiyonlar, abs() gibi tek bir isimle adlandırılırlar. Fonksiyonu çağırmak için kullanılan veri tipi, gerçekte hangi fonksiyonun çalışacağını belirler. Böylece bir fonksiyon adının birkaç farklı amaç için kullanılması mümkündür. Buna fonksiyonların aşırı yüklenmesi denir.

Polimorfizm-1

```
        class Printdata
        {
            void print(int i)
            {
                Console.WriteLine("Printing int: {0}", i);
            }

            void print(double f)
            {
                Console.WriteLine("Printing float: {0}", f);
            }
```

```
        void print(string s)
        {
            Console.WriteLine("Printing string: {0}", s);
        }
        static void Main(string[] args)
        {
            Printdata p = new Printdata();

            // Call print to print integer
            p.print(5);

            // Call print to print float
            p.print(500.263);

            // Call print to print string
            p.print("Hello C++");
            Console.ReadKey();
        }
    }
```

Polimorfizm - 2

```
public class DrawingObject
{
    public virtual void Draw()
    {
        Console.WriteLine("I'm just a generic drawing object.");
    }
}


public class Line : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("I'm a Line.");
    }
}

public class Circle : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("I'm a Circle.");
    }
}

public class Square : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("I'm a Square.");
    }
}


public class DrawDemo
{
```

```
        public static int Main()
        {
            DrawingObject[] dObj = new DrawingObject[4];

            dObj[0] = new Line();
            dObj[1] = new Circle();
            dObj[2] = new Square();
            dObj[3] = new DrawingObject();

            foreach (DrawingObject drawObj in dObj)
            {
                drawObj.Draw();
            }

            return 0;
        }
    }
```

Inheritance

- Kalıtım yolu ile eldeki sınıflardan yeni sınıflar türetilir.
- Türeyen sınıflar türedikleri sınıfın özelliklerini kalıtım yoluyla devralırlar ve kendisi de yeni özellikler tanımlayabilir.
- Türetme ile sınıflar arasında hiyerarşik bir yapı kurulabilir.

Kod Kalıtımı: Eğer yeni bir sınıf tanımlıyorsanız ve mevcut bir sınıfın işlevselliğinden yararlanmak istiyorsanız, yeni sınıfınızı mevcut sınıftan türetirsiniz.

Arayüz Kalıtımı: Bir diğer kalıtım stratejisi, türetilmiş sınıfın üst sınıfının eleman fonksiyonlarının yalnızca isimlerini kalıtım yoluyla almasıdır. Türetilmiş sınıf bu fonksiyonlar için kendi kodunu kullanır. Arayüz kalıtımının temel faydası çok biçimliliğe izin vermesidir.

```
   using System;
namespace InheritanceApplication
    {
        class Shape
        {
            public void setWidth(int w)
            {
                width = w;
            }
            public void setHeight(int h)
            {
                height = h;
            }
            protected int width;
            protected int height;
        }

        // Derived class
        class Rectangle : Shape
        {
            public int getArea()
            {
                return (width * height);
            }
        }
```

```
        class RectangleTester
        {
            static void Main(string[] args)
            {
                Rectangle Rect = new Rectangle();

                Rect.setWidth(5);
                Rect.setHeight(7);

                // Print the area of the object.
                Console.WriteLine("Total area: {0}", Rect.getArea());
                Console.ReadKey();
            }
        }
    }
```

Abstraction

Nesnenin bilinmesi gerekmeyen detaylarının gizlenmesi anlamına gelir. Örneğin bir araba kullanıcısı araba kullanırken kontağı çalıştırır, frene, gaza basar veya direksiyon çevirir. Yani araba nesnesinin kendisini ilgilendiren ve değiştirebileceği kısımlarını değiştirir. Araba nesnesinin motorunun nasıl çalıştırdığına veya benzini nasıl aldığını bilmesine gerek yoktur.

Benzer şekilde kod yazılırken de, nesnelerin karmaşıklıkları diğer nesnelerden gizlenmelidir. O nesnenin içinde neler olup bittiğini bilmeye ihtiyaç duymadan sadece nasıl kullanılacağını bilmek yeterli olmalıdır.

Bunu sağlamak için nesneler için bir soyut sınıf yazılır ve ortak özellikler burada toplanır. Bu soyut sınıftan türeyen farklı nesnelerin birbirlerinin içinde olup bitenden haberi olması gerekmez.

(Virtual metod: türetilmiş sınıfında içeriği override edilerek değiştirilebilen metodlardır.)

```
    public abstract class Sebze
    {
        public virtual bool SulamaGerekMi(float AraziSicakligi, float HavaNemi,
float AraziNemi) { return false; }

        public void Sula()
        {
            //Burada sulama cihazi ile bir miktar su verme islemi yapilir.
        }
    }
    internal class Domates : Sebze
    {
        public override bool SulamaGerekMi(float AraziSicakligi, float HavaNemi,
float AraziNemi)
        {
            //Burada parametreler kontrol edilir, ve gerekli sonuc dondurulur
            return true;
        }
    }
    internal class Lahana : Sebze
    {
        public override bool SulamaGerekMi(float AraziSicakligi, float HavaNemi,
float AraziNemi)
        {
            //Burada parametreler kontrol edilir, ve gerekli sonuc dondurulur
            return true;
```

```
        }
    }
    internal sealed class IrrigationSystem
    {
        List<Sebze> sebzeler = new List<Sebze>();
        public void SulamaIslemi()
        {
            //Burada gerekli duzenlemelerin yapildigi varsayilsin

            float araziSicakligi = 0;
            float havaNemi = 0;
            float araziNemi = 0;
            foreach (Sebze item in sebzeler)
            {
                //Sensörlerden gelen veriler okunur ve fonksiyona verilir
                if (item.SulamaGerekMi(araziSicakligi, havaNemi, araziNemi))
                {
                    item.Sula();
                }
            }
        }
    }
```

Dikkat:

Encapsulation işleminde amaç, nesnenin verilerini güvene almak. Erişimi kontrol etmekti. Soyutlamada ise nesnenin iç yapısı yani karmaşıklığı bu nesne ile etkileşime geçen diğer nesnelerden uzak tutulur.

Ödev 3. Abstraction ile Encapsulation farklarını araştırınız.

Interface

Sınıf değildir.

Interface yani arayüz, sınıflar için ortak bir arayüz sağlarlar. Ortak bir anlaşma metni gibidir.

Diğer bir deyişle bu arayüzü kullanan sınıfların, arayüz tarafından belirlenen belli özellikleri sağladığını gösterir.

Interface sınıflarda definition(içerik) olmaz sadece declare (bildirim) yapılır.

Interface ile çoklu kalıtım yapılabilir, yani bir sınıf birden fazla interface' i sağlayabilir.

Interface içinde içerik tanımlaması yapılmadığından, interface ile bu interface' i sağlayan sınıf arasındaki bağımlılık oldukça azdır.

Abstract Sınıf

Sınıf yapısının özel bir türüdür.

Abstract sınıf örneklenemez. Diğer sınıflar abstract sınıftan türetilir.

Soyut sınıflardır. Sınıfları abstract olarak tanımlayarak, abstraction ve polymorphism' den faydalanılabilir.

Abstract sınıflar ortak davranışı olan sınıflar için kullanılır.

Abstract sınıflarda, (definition) tanımlama kısmen yapılır.

Genellikle bir kısım yerler implement edilirken (ortak kısımlar), bazı kısımlar virtual olarak bırakılır. Böylece miras alan nesneler bu kısımları kendine göre override edebilir.

Yani hem ortak bir anlaşma, hem de ortak bir düzenin varlığı durumunda faydalanılır.

Abstract sınıf ile çoklu kalıtım yapılmaz.

Ortak kodların tekrar yazımının azaltılması için faydalanılır.

Ancak bu özelliğinden dolayı abstract sınıf ile onu miras alan sınıf arasındaki bağımlılık Interface' e göre daha yüksektir.

Ödev 4: Abstract method nedir? Virtual method' dan farkı nedir?