11-ші дәріс. Кластарды мұралау

- **Кластар иерархияларын (сатыларын) ұйымдастыру;**
- **Алдыңғы және соңғы байланыстыру;**
- **Виртуалды тәсілдер;**
- **Абстрактілі және туындысыз кластар;**
- **Кластар арасындағы өзара қатынас түрлері.**

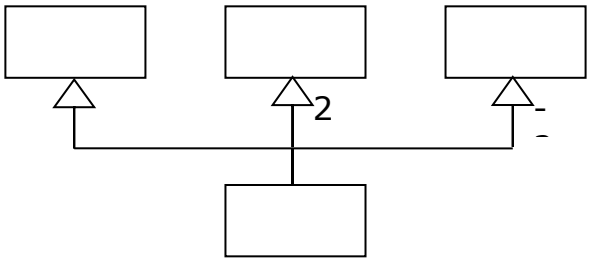
Мұралау мүмкіндіктері

- Мұралау ОБП-дың қуатты құралы болып табылады. Ол ұрпақ-кластардың ата-кластар қасиеттерін иемденетін және оларды толықтыру немесе өзгерту мүмкіндігіне ие болатын иерархияларды құруға мүмкіндік береді.
- Мұралау төмендегідей өзара байланысты мақсаттар үшін қолданылады:
- программадан қайталанатын код фрагменттерін жою;
- программаны өзгертуді жеңілдету;
- бұрын құрылған программалар негізінде жаңа программалар құруды жеңілдету.
- Сонымен қатар, программалардың бастапқы кодын пайдалануға мүмкіндік болмайтын жағдайда, оларға өзгеріс енгізуді қажет ететін объектілерді қолданудың жалғыз мүмкіндігі – осы мұралау болып табылады.

Синтаксисі (жазылуы):

[атрибуттар] [спецификаторлар] class класс_аты [:ата_тектері]
класс тұлғасы

```
class Monster
    {   ... // private және public-тен басқа,
              // protected қолданылады
    }
        class Daemon : Monster
    {     ...
    }
```



- C# тілінде кластың ұрпақтары саны бірнешеу болуы мүмкін
- Кластың тек бір ата-кластан және интерфейстердің кез келген санынан мұралау мүмкіндігі бар.
- Мұралау кезінде ұрпағы ата-тегінің барлық элементтерін қабылдайды.
- private элементтерін оның ұрпақтары тікелей пайдалана алмайды.
- protected элементтерін тек ұрпақтары пайдалана алады.

Кластың жалпы мысалы

```
class Monster {
      public Monster()          // конструктор
      {
                this.name    = "Noname";
                this.health = 100;
                this.ammo = 100;
      }
   public Monster( string name ) : this()
      {         this.name = name;         }
   public Monster( int health, int ammo, string name )
       {        this.name    = name;
                this.health = health;
                this.ammo = ammo;
```

```
        }
  public int Health {                  // қасиет
     get { return health;    }
     set { if (value > 0) health = value;
                  else                          health = 0; }
  }
public int Ammo {                  // қасиет
     get { return ammo;    }
     set { if (value > 0) ammo = value;
                  else                          ammo = 0; }
  }
  public string Name {                  // қасиет
     get { return name;    }
}
  public void Passport()                          // тәсіл
        {      Console.WriteLine(
    "Monster {0} \t health = {1} \
       ammo = {2}", name, health, ammo );
          }
public override string ToString(){
      string buf = string.Format(
           "Monster {0} \t health = {1} \
        ammo = {2}", name, health, ammo);
       return buf;    }
string name;           // private өрістер
   int health, ammo;
Daemon, Monster класының мұрагері
class Daemon : Monster {
        public Daemon() {      brain = 1;          }
        public Daemon( string name, int brain ) : base( name )    this.brain = brain;      }
        public Daemon( int health, int ammo, string name, int brain )
                  : base( health, ammo, name )        {    this.brain = brain;          }
new public void Passport() {
                Console.WriteLine( "Daemon {0} \t health ={1} ammo ={2} brain ={3}",
Name, Health, Ammo, brain );
             }
        public void Think()
                {    Console.Write( Name + " is" );
                   for ( int i = 0; i < brain; ++i )
                            Console.Write( " thinking" );
                   Console.WriteLine( "..." );
                }
int brain;              // жабық өріс
}
public void Passport()                  // тәсіл
        {
Console.WriteLine(
                "Monster {0} \t health = {1} \
                ammo = {2}",
                  name, health, ammo );
```

```
    }
```

Конструкторлар және мұралау

Конструкторлар мұраланбайды, сондықтан туынды кластың өзінің конструкторлары (программалаушы немесе жүйе құрған) болуы тиіс.

Конструкторларды шақыру реттілігі:

- Егер туынды класс конструкторында базалық класс конструкторын нақты шақыру көрсетілмесе, автоматты түрде базалық класс конструкторы параметрлерсіз түрде шақырылады.
- Бірнеше деңгейден тұратын иерархия үшін базалық кластар конструкторлары ең жоғарғы деңгейден бастап шақырылады. Осыдан кейін объект болып табылатын класс элементтерінің конструкторлары олардың класта жариялану реті бойынша, содан кейін класс конструкторы орындалады.
- Егер базалық класс конструкторы параметрлердің көрсетілуін талап етсе, онда ол туынды класс конструкторында инициалдау тізімінде.

Базалық класс конструкторын шақыру

```
public Daemon( string name, int brain ) : base( name )              // 1
        {
            this.brain = brain;
        }
public Daemon( int health, int ammo, string name, int brain )
            : base( health, ammo, name )                                    // 2
        {
            this.brain = brain;
        }
```

Өрістер мен тәсілдерді мұралау

- **Кластың өрістері, тәсілдері және қасиеттері мұраланады.**
- **Базалық класс элементін жаңа элементке алмастыру қажет болғанда new түйінді сөзін қолданған жөн:**

```
// Daemon класының тәсілдері (тегінің функцияларын толықтыру)
new public void Passport()
{
base.Passport();        // ата-тегінің функцияларын қолдану
        Console.WriteLine( brain );             // толықтыру
}
```

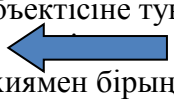
```
// Daemon класының тәсілі (толық алмастыру)
new public void Passport() {
            Console.WriteLine( "Daemon {0} \t health ={1} ammo ={2} brain
={3}",
Name, Health, Ammo, brain );
            }

// Monster класының тәсілі
public void Passport()
      {
        Console.WriteLine(
            "Monster {0} \t health = {1} \
```

```
                ammo = {2}",
name, health, ammo );
        }
```

Мұралау кезіндегі типтер үйлесімділігі

Базалық класс объектісіне туынды класс объектісін меншіктеуге болады:



ұрпағы

Бұл бүкіл иерархиямен бірыңғай жұмыс жасау үшін орындалады.

Программаны бастапқы кодтан орындалатын кодқа түрлендіру кезінде екі байланыстыру механизмі қолданылады:

- Алдыңғы (раннее) – early binding – программа орындалғанға дейін
- Соңғы (позднее - динамикалық) – late binding – орындалуы кезінде

Алдыңғы байланыстыру мысалы

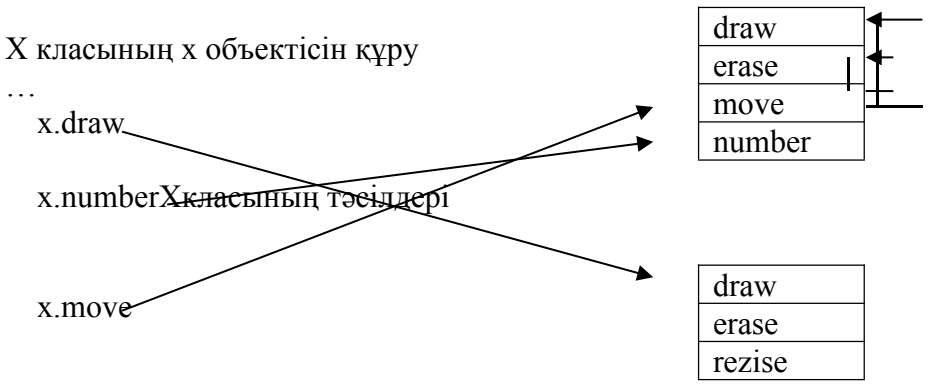
```
class T {
        T(int i) { this.i = i; }
                draw {"TT" шығару}
        erase
        move { erase, ->, draw }
                number {i шығару}
protected int i;
}
class X : T {
        X(int i) { this.i = i; }
new     draw {"XX“ шығару}
new     erase
        resize
}
// барлық public тәсілдері

// жалғыз объект:
X x = new X(15);
x.draw();                                       // XX
x.number();                                     // 15
x.move()                                        // TT
// базалық типтегі объектілер жиымы:
T mas = new T[n];
mas[0] = new T(10);
mas[1] = new T(20);
mas[2] = new X(15);
mas[3] = new X(25);
foreach (T t in mas) t.number()
// 10      20      15      25
foreach (T t in mas) t.draw()
// TT      TT      TT      TT
// t.resize – орындалмайды
```

Алдыңғы байланыстыру

Шақыратын тәсіл

X класының x объектісін құру



Алдыңғы байланыстыру

- Сілтемелерге программа орындалғанға дейін рұқсат беріледі
- Сондықтан компилятордың тек тәсіл немесе қасиет шақырылатын айнымалының типін басшылыққа алу мүмкіндігі бар. Бұл айнымалыда әртүрлі уақытта әртүрлі типтегі объектілерге сілтемелер болуының мүмкіндігін компилятор есепке алмайды.
- Сондықтан туынды типті объект меншіктелген базалық типті сілтеме үшін тек базалық класта анықталған тәсілдер мен қасиеттерді шақыруға болады (яғни, класс объектілеріне қатынас құру мүмкіндігі сілтеме вариантйтын объект типімен емес, сілтеме типімен анықталады).

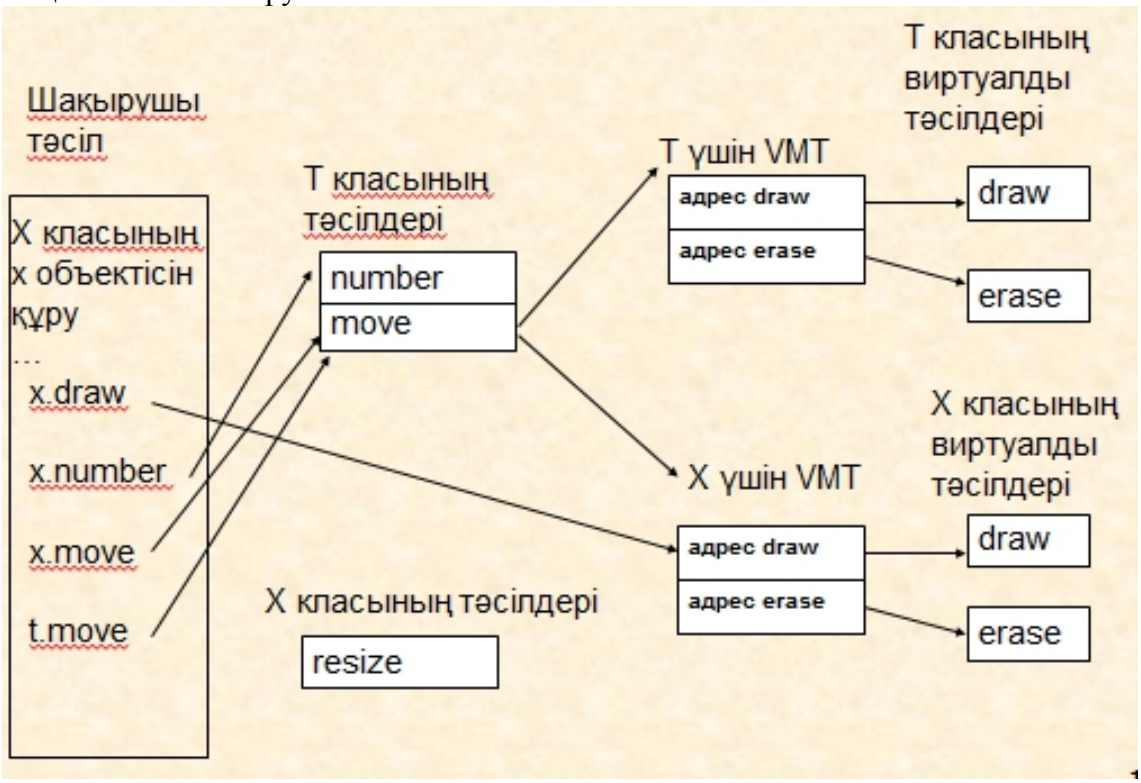
Соңғы байланыстыру мысалы

```
class T {
        T(int i)
virtual draw { "TT" }
virtual erase
        move { erase, …, draw }
                number { i }
protected int i;
}
class X : T {
                X(int i)
override draw { "XX" }
override erase
                resize
}
// барлық public тәсілдері

        // жеке объект:
        X x = new X(15);
        x.draw();                               // XX
        x.number();                          // 15
        x.move()                                // XX
        // баз. типті объектілер жиымы:
        T mas = new T[n];
        mas[0] = new T(10);
```

```
mas[1] = new T(20);
mas[2] = new X(15);
mas[3] = new X(25);
foreach (T t in mas) t.number()
// 10      20      15      25
foreach (T t in mas) t.draw()
// TT      TT      XX      XX
```

Соңғы байланыстыру



Полиморфизм

- *Базалық кластың виртуалды тәсілдері иерархияның түгелдей интерфейсін анықтайды.*
- Ол ұрпақтарда жаңа виртуалды тәсілдер қосу арқылы кеңейтіле алады. Виртуалды тәсілді ұрпақтардың әрбіреуінде қайта анықтаудың қажеті жоқ: егер тәсіл ұрпаққа қажетті әрекеттерді орындайтын болса, ол мұраланады.
- Виртуалды тәсілді шақыру келесідей орындалады: объектіден оның виртуалды тәсілдер кестесі алынады, оның ішінен тәсілдің адресі таңдалып, басқару осы тәсілге беріледі.
- Осылайша, виртуалды тәсілдерді қолдану кезінде иерархияның барлық аттас тәсілдерінің ішінен оны шақырған объектінің типіне сәйкес келетін тәсіл таңдалады.
- Виртуалды тәсілдердің көмегімен объектіге бағытталған программалаудың негізгі принциптерінің бірі – полиморфизм жүзеге асырылады.

Виртуалды тәсілдерді қолдану

- Виртуалды тәсілдер туынды кластармен базалық класқа сілтеме жасау арқылы жұмыс істегенде қолданылады.
- Сонымен қатар, объектілерді тәсілдерге параметр ретінде беру кезінде виртуалды тәсілдер қолданылады. Тәсілдің параметрлерінде базалық типті объект сипатталады, ал шақыру кезінде оған туынды класс объектісі беріледі. Бұл жағдайда тәсілдің ішіндегі объект үшін шақырылатын виртуалды тәсілдер параметрдің емес, аргументтің типіне сәйкес болады.
- Кластарды сипаттау кезінде виртуалды тәсілдер ретінде туынды кластарда басқа жолмен орындалатын тәсілдерді анықтау ұсынылады. Егер иерархияның барлық кластарында тәсіл бірыңғай жүзеге асатын болса, оны жай тәсіл ретінде анықтаған жөн.

Абстрактілі кластар

- Абстрактілі класс тек ұрпақтарды тудыру үшін қолданылады. Әдетте онда ұрпақтардың әрқайсысы өзінің қолданатын жолымен жүзеге асыратын тәсілдер жиыны беріледі. Абстрактілі кластар туынды кластарда нақтылануы жоспарланатын жалпы ұғымдарды бейнелеуге арналған.
- Абстрактілі класс иерархия үшін толықтай интерфейсті анықтайды және бұл кезде класс тәсілдеріне ешбір нақты әрекеттер сәйкес келмеуі мүмкін. Бұл жағдайда тәсілдердің тұлғасы бос болады және олар abstract спецификациясымен жарияланады.
- Егер класта кем дегенде бір абстрактілі тәсіл бар болса, класс түгелдей абстрактілі класс ретінде (abstract спецификаторымен) сипатталуы тиіс.
- Абстрактілі кластың құрамында толықтай анықталған тәсілдер де болуы мүмкін, бұл оның интерфейстен айырмашылығы.

Полиморфты тәсілдер

- Абстрактілі кластар келесі жағдайларда қолданылады:
- бір иерархияның объектілерін сақтауға арналған мәліметтер құрылымдарымен жұмыс істеу кезінде
- тәсілдердің параметрлері ретінде.
- Егер абстрактілі кластан туындаған класс барлық абстрактілі тәсілдерді қайта анықтайтын болмаса, ол да абстрактілі класс ретінде сипатталуы тиіс.
- Параметрі абстрактілі класс болатындай тәсіл құруға мүмкіндігі бар. Программаның орындалуы барысында бұл параметрдің орнына кез келген туынды кластың объектісі берілуі мүмкін. Бұл бір иерархия шеңберіндегі кез келген типті объектімен жұмыс істейтін полиморфты тәсілдерді құруға мүмкіндік береді.

Жүзеге асырылған тәсілге интерфейс типті объект арқылы қатынас құру

- Бұл жолдың ыңғайлылығы IAction типті объектілерге осы интерфейсті қолдайтын әртүрлі класс объектілеріне сілтемені меншіктеу кезінде байқалады.

- Мысалы, интерфейс типті параметрі бар тәсіл бар болсын. Бұл параметрдің орнына интерфейсті жүзеге асыратын кез келген объектіні беруге болады:

```
static void Act( IAction A )
{
      A.Draw();
}
static void Main()
{
     Monster Vasia = new Monster( 50, 50, "Вася" );
     Act( Vasia );
     ...
}
```

Интерфейсті жүзеге асырудың екінші тәсілі

Жүзеге асырылатын элементтің алдында *интерфейс атын нақты көрсету.*
Қатынас құру спецификаторлары көрсетілмейді. Мұндай элементтерге программада тек *интерфейс типті объект арқылы* қатынас құруға болады:

```
     class   Monster :   IAction {
          int     IAction.Power { get{ return ammo * health;}}
          void IAction.Draw() {
               Console.WriteLine(" Мұнда    " + name + " болды");       }                    }
...
IAction    Actor = new Monster( 10, 10, "Маша" );
Actor.Draw();                                  // интерфейс типті объект арқылы қатынас құру
// Monster Vasia = new Monster( 50, 50, "Вася" );
// Vasia.Draw();                                                                  қате!
```

Мұндай жағдайларда бұларға сәйкес тәсіл *класс интерфейсінің құрамына кірмейді*. Бұл оны интерфейстің қандай да бір элементтері кластың соңғы қолданушысына қажет емес болғанда ғана оны жеңілдетуге мүмкіндік береді.
Оның үстіне, бұл тәсіл көптік мұралау кезінде туындайтын қарама-қайшылықтарды болдырмауға да мүмкіндік береді.

Мысал

Monster класы екі интерфейсті сүйемелдейді делік: біреуі объектілерді басқару үшін, ал екіншісі тестілеу үшін қолданылады:

```
interface Itest { void Draw(); }
interface Iaction { void Draw(); int Attack( int a ); … }
class Monster : IAction, Itest {
     void ITest.Draw()    {
               Console.WriteLine( "Testing " + name );       }
     void IAction.Draw() {
               Console.WriteLine(" Мұнда " + name + " болды ");       }
     ... }
```

Екі интерфейсте де сигнатуралары бірдей болып келетін Draw тәсілі бар. Оларды ажырату үшін интерфейс атын нақты түрде көрсету көмек береді.
Осы тәсілдерге типті келтіру операциясын қолдану арқылы қатынас құру былай орындалады:

```
Monster Vasia = new Monster( 50, 50, "Вася" );
((ITest)Vasia).Draw();                                  // нәтижесі: Мұнда Вася болды
```

((IAction)Vasia).Draw(); // нәтижесі: Testing Вася

is операциясы

- Объектімен интерфейс типті объект арқылы жұмыс істеу кезінде объект берілген интерфейсті сүйемелдейтініне көз жеткізу керек.
- Тексеру is бинарлық операциясының көмегімен орын-далады. Ол is түйінді сөзінің сол жағында орналасқан объектінің ағымдағы типі оның оң жағында берілген типпен сәйкес келетінін немесе сәйкес келмейтінін анықтайды.
- Егер объектіні берілген типке түрлендіруге мүмкіндік болса, операция нәтижесі true, кері жағдайда false болады. Әдетте, операция келесі контекстіде қолданылады:

```
if ( объект is тип )
{
    // "объектіні" "типке" түрлендіруді орындау
    // түрлендірілген объектімен әрекеттер орындау
}
```

as операциясы

- as операциясы берілген типке түрлендіруді орындайды, ал егер бұл мүмкін болмаса, онда ол null нәтижесін қалыптастырады:

```
static void Act( object A )
{
    IAction Actor = A as IAction;
    if ( Actor != null ) Actor.Draw();
}
```

Қарастырылған операциялардың екеуі де интерфейс-терге де, кластарға да қолданыла береді

Интерфейстер және мұралау

- Интерфейстің ата-тегі болмауы мүмкін немесе олардың саны бірнешеу болуы да мүмкін, соңғы жағдайда ол ең жоғарғы деңгейден бастап, өзінің барлық базалық интерфейстерінің барлық элементтерін мұралайды.
- Базалық интерфейстерге олардың ұрпақтарынан кем болмайтындай деңгейде қол жеткізілетін болуы тиіс.
- Әдеттегі кластар иерархиясы сияқты, базалық интер-фейстер жалпылама әрекеттер сипатын анықтайды, ал олардың ұрпақтары оны нақтылайды және толықтырады.
- Сонымен қатар, ұрпақтар интерфейсінде сигнатурасы бірдей, мұраланған элементтерді қайта анықтаушы элементтерді көрсетуге болады. Мұндайда элементтің алдына кластардағы сияқты new түйінді сөзі жазылады. Осы сөздің көмегімен базалық интерфейстің соларға сәйкес элементі жасырылады.

Мысал

```
interface IBase { void F( int i ); }
interface Ileft : IBase {
    new void F( int i );                /* F тәсілін қайта анықтау */ }
```

```
interface Iright : IBase    {      void G();    }
interface Iderived : ILeft, IRight {}
class A    {
     void Test( IDerived d ) {
          d.F( 1 );                                   // ILeft.F шақырылады
          ((IBase)d).F( 1 );                  // IBase.F шақырылады
          ((ILeft)d).F( 1 );                   // ILeft.F шақырылады
          ((IRight)d).F( 1 );                 // IBase.F шақырылады
     }
}
```

IDerived — IRight — IBase тізбегінде қайта анықталмағанына қарамастан, IBase интерфейсінің F тәсілі ILeft интерфейсімен жасырылған.

Интерфейстерді жүзеге асырудың ерекшеліктері

- Интерфейсті жүзеге асыратын класс оның барлық элементтерін, оның ішінде, мұраланғандарын да анықтауы керек. Егер осындай жағдайда интерфейстің аты нақты түрде көрсетілетін болса, ол осыған сәйкес элемент сипатталған интерфейске сілтеме жасауы тиіс.
- Өзіндік немесе мұраланған элементтерге нақты сілтеме жасайтын интерфейс класс ата-тектері тізімінде көрсетілуі тиіс.
- Класс өзінің ата тегінің барлық тәсілдерін, оның ішінде, интерфейстерді жүзеге асырғандарын да мұралайды. Ол осы тәсілдерді new спецификаторының көмегімен қайта анықтай алады, алайда олармен тек класс объектісі арқылы ғана қатынас құруға болады.

Стандартты .NET интерфейстері

- .NET кластар кітаханасында объектілердің қажетті әрекет-терін тағайындайтын көптеген стандартты интерфейстер анықталған. Мысалы, IComparable интерфейсі объекті-лерді «үлкен-кіші» деген сияқты салыстыру тәсілін тағайындайды, бұл оларды реттеуге мүмкіндік береді.
- IEnumerable және IEnumerator интерфейстерін жүзеге асыру foreach көмегімен объект құрамын көруге, ал ICloneable интерфейсін жүзеге асыру объектілерді клондауға мүмкіндік береді.
- Стандартты интерфейстерді кітапхананың көптеген стандартты кластары сүйемелдейді. Мысалы, foreach көмегімен жиыммен жұмыс істеу мүмкіндігінің болу себебі – Array типі IEnumerable және IEnumerator интерфейс-терін жүзеге асырады.
- Стандартты интерфейстерді сүйемелдейтін өз кластары-мызды да құруға болады, бұл осы кластар объектілерін стандартты тәсілдер көмегімен пайдалануға мүмкіндік береді.

Объектілерді салыстыру

- **IComparable** интерфейсі **System** атаулар кеңістігінде анықталған. Оның құрамында тек бір ғана **CompareTo** тәсілі бар, ол екі объектіні – ағымдағы және оған параметр ретінде берілген объектілерді салыстыру нәтижесін қайтарады.

```
interface IComparable
{
```

```
    int CompareTo( object obj )
}
```

- Тәсіл төмендегідей нәтижелерді қайтаруы тиіс:
  - 0, егер ағымдағы объект пен параметр тең болса;
  - теріс сан, егер ағымдағы объект параметрден кіші болса;
  - оң сан, егер ағымдағы объект параметрден үлкен болса.

Интерфейсті жүзеге асыру мысалы

- sealed түйінді сөзі абстрактілі класқа қарама-қарсы, яғни мұралауға тиым салынатын класты сипаттауға мүмкіндік береді:

```
sealed class Spirit      {   ...      }
//  class Monster : Spirit { ... }                  қате!
```

- Құрамдас мәліметтер типтерінің басым бөлігі sealed ретінде сипатталған. Егер туындысыз кластың мүмкіндіктерін пайдалану қажет болса, мұралау емес, *кіріктіру (вложение)* немесе *қосу (включение)* қолданылады: кластың ішінде сәйкес типті өріс сипатталады.
- Әдетте класс өрістері жабық болатындықтан, қосылған класс тәсілі шақырылатын қамтушы класс тәсілін сипаттайды. Кластардың өзара байланысының мұндай тәсілі *қосу-делегирлеу (ұсыну) моделі* деген атпен белгілі *(бұл жайлы – 25 слайд)*.

object класы

- C# тілінде object деп аталатын .NET объектілер иерархиясының System.Object түпкі класы барлық ұрпақтарды бірнеше маңызды тәсілдермен қамтамасыз етеді.
- Туынды кластар бұл тәсілдерді тікелей қолдана алады немесе оларды қайта анықтай алады.
- object класы келесі жағдайларда тікелей қолданылады:
  - тәсілдер параметрлеріне ортақтық сипат беру үшін олардың типін сипаттау кезінде;

System.Object класының ашық тәсілдері

```
public virtual bool Equals(object obj);
```

- егер параметр мен шақырушы объект бір жады аймағына сілтеме жасаса, true мәнін қайтарады

```
public static bool Equals(object ob1, object ob2);
```

- егер параметрлердің екеуі де бір жады аймағына сілтеме жасаса, true мәнін қайтарады

```
public virtual int GetHashCode();
```

- объектінің хэш-кодын қалыптастырады және объектіні бірмәнді идентификациялайтын санды қайтарады

```
public Type GetType();
```

- объектінің ағымдық полиморфты типін қайтарады (сілтеме типін емес, сол сілтеме қазіргі уақытта көрсетіп тұрған объект типін)

```
public static bool ReferenceEquals(object ob1, object ob2);
```

- егер параметрлердің екеуі де бір жады аймағына сілтеме жасаса, true мәнін қайтарады

public virtual string ToString()

- сілтемелік кластар үшін кластың толық атын қатар түрінде, ал мәндік кластар үшін қатарға түрлендірілген шаманың мәнін қайтарады. Объектінің күйі туралы ақпаратты шығару үшін бұл тәсілді қайта анықтайды.

Equals тәсілін қайта анықтау мысалы

```
// сілтемелерді емес, мәндерді салыстыру
public override bool Equals( object obj ) {
        if ( obj == null || GetType() != obj.GetType() ) return false;
        Monster temp = (Monster) obj;
        return    health == temp.health &&
                        ammo      == temp.ammo      &&
                        name      == temp.name;
}
public override int GetHashCode()
{
        return name.GetHashCode();
}
```

Программалау үшін берілетін ұсыныстар

- Мұралаудың негізгі артықшылығы – базалық класс деңгейінде туынды класс объектілерімен де жұмыс істеуге мүмкіндік беретін универсалды код жазуға болады, бұл виртуалды тәсілдер көмегімен жүзеге асырылады.
- Виртуалды тәсілдер ретінде иерархияның барлық кластарында бір функцияны (мүмкін әртүрлі тәсілдермен) атқаратын тәсілдер сипатталуы тиіс.
- Туынды кластарда нақтылануы жоспарланатын жалпы ұғымдарды бейнелеу үшін абстрактілі кластар қолданылады. Әдетте абстрактілі класта ұрпақтардың әрқайсысы өзінің қолданатын жолымен жүзеге асыратын тәсілдер жиыны, яғни интерфейс беріледі.
- Жай тәсілдерді (виртуалды емес) туынды класта қайта анықтау ұсынылмайды.

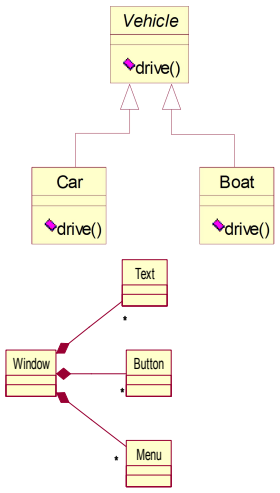
Кластар арасындағы өзара байланыс түрлері

- **Мұралау**
  - Специализация (Ұрпақ өз ата тегінің арнайы формасы болып табылады)
  - Спецификация (Ұрпақ-класс тегінде сипатталған әрекеттерді жүзеге асырады)
  - Конструирлеу или Вариациялау (Мұрагер тегінің тәсілдерін қолданады, алайда оның ішкі типі болып табылмайды; ұрпақ пен тегі бір тақырыптың вариациялары болып табылады – мысалы, тіктөртбұрыш пен квадрат)
  - Кеңейту (Ұрпаққа тегінің әрекеттерін кеңейтетін жаңа тәсілдер қосылады)
  - Жалпылау (Ұрпақ тегінің әрекетін жалпылайды)
  - Шектеу (Ұрпақ тегінің әрекетін шектейді)

- Кіріктіру
  - композиция
  - агрегация

  - Специализация (Ұрпақ өз ата тегінің арнайы формасы болып табылады)
  - Спецификация (Ұрпақ-класс тегінде сипатталған әрекеттерді жүзеге асырады)
  - Конструирлеу или Вариациялау (Мұрагер тегінің тәсілдерін қолданады, алайда оның ішкі типі болып табылмайды; ұрпақ пен тегі бір тақырыптың вариациялары болып табылады – мысалы, тіктөртбұрыш пен квадрат)
  - Кеңейту (Ұрпаққа тегінің әрекеттерін кеңейтетін жаңа тәсілдер қосылады)
  - Жалпылау (Ұрпақ тегінің әрекетін жалпылайды)
  - Шектеу (Ұрпақ тегінің әрекетін шектейді)
- Кіріктіру
  - композиция
  - агрегация

Жүзеге асырылған тәсілге интерфейс типті объект арқылы қатынас құру



- Y *класының* X *класынан мұралауы* басым жағдайда Y класы X класының бір түрі (нақтырақ, жеке концепциясы) болып табылатынын білдіреді.

- *Кіріктіру(вложение)* бір кластың екінші класты пайдалануының мұралаудан өзгеше механизмі болып табылады:бір класс екіншісінің өрісі болады.

- *Кіріктіру* «Y құрамында X бар» немесе «Y X арқылы орындалады» деген класс байланыстарын бейнелейді немесе «қосу-делегирлеу» моделінің көмегімен жүзеге асырылады.