5 дәріс. Әдістерді асыра жүктеу. Статикалық кластар және кластың статикалық мүшелері.

Дәрістің мақсаты: студенттерде әдістерді асыра жүктеудің қызметі және кластың статикалық мүшелерін пайдалану ерекшеліктері туралы түсініктерін көрсетуге қабілет қалыптастыру.

Осы дәрісті меңгеру нәтижесінде студенттер келесі қабілеттерге ие болады:

- Әдістерді асыра жүктеудің қызметі туралы түсініктерін көрсету;
- Әдістерді асыра жүктеу шарттарын көрсету;
- Кластың статикалық мүшелерін құру және пайдалану ерекшеліктері туралы түсініктерін көрсету.

Бір класс ішіндегі *бірнеше әдістердің аттары бірдей болып*, тек *олардың параметрлері әртүрлі болып жариялануы әдістердің асыра жүктелуі* деп аталады. C# тіліндегі *әдістердің асыра жүктелуі* ОБП-дағы полиморфизмді жүзеге асырудың бір жолы болып табылады.

Жалпы, әдістерді асыра жүктеу үшін олардың әртүрлі нұсқаларын жариялау жеткілікті, ал қалған жағын компилятор орындайды. Мұнда мынадай маңызды шартты орындау керек: асыра жүктелетін әрбір әдістің параметрлерінің типтері немесе олардың сандары әртүрлі болуы керек. Екі әдістің тек қайтаратын мәндерінің типтерінің әртүрлі болуы жеткіліксіз. Олар өз параметрлерінің типтерімен немесе сандарымен өзгеше болуы керек.

Әрине, асыра жүктелген әдістердің қайтарылатын типтері де әртүрлі бола береді, бірақ олар қай әдісті таңдап алып, орындау керектігі жайлы толық мәлімет бере алмайды. Асыра жүктелген әдіс шақырылғанда, оның параметрлерінің әдіске берілетін аргументтерге сәйкес келген (типтері және сандары арқылы) нұсқасы орындалады.

Төменде әдістерді асыра жүктеуді көрсететін қарапайым мысал берілген.

Мысал 1. Әдістерді асыра жүктеуді көрсету.

```
using System;
class Tortburysh {
  public double a;
  public double b;
  public Tortburysh(double a, double b)
  {  this.a=a; this.b = b; }
    public void Shygaru() {
      Console.WriteLine("tortburysh qabyrgalary: "+a+", "+b);
    }
    // Бір бүтін параметрлі eseleu әдісін  асыра жүктеу.
    public Tortburysh eseleu(int k) {
      return new Tortburysh(a*k, b*k);
    }
    // Бір нақты параметрлі eseleu әдісін  асыра жүктеу.
    public Tortburysh eseleu(double k) {
      return new Tortburysh(a*k, b*k);
    }
    // Екі бүтін параметрлі eseleu әдісін  асыра жүктеу.
    public Tortburysh eseleu(int k, int i) {
      return new Tortburysh(a*k, b*i);
  }
}
class Program  {
```

```
  static void Main() {
    Tortburysh ob1 = new Tortburysh (4,5);
      Console.WriteLine("bastapqy tortburysh: ");
      ob1.Shygaru();
      Console.WriteLine("eseleu adisin butin parametrmen
                  shaqyru");
      Tortburysh ob2 = ob1.eseleu(2);
    ob2.Shygaru();
    Console.WriteLine("eseleu adisin naqty parametrmen
                shaqyru");
      Tortburysh ob3 = ob1.eseleu(2.3);
      ob3.Shygaru();
      Console.WriteLine("eseleu adisin eki butin parametrmen
                  shaqyru");
      Tortburysh ob4 = ob1.eseleu(2,3);
      ob4.Shygaru();
  }
}
```

Жоғарыдағы программа жұмысының нәтижесі:

```
bastapqy tortburysh:
tortburysh qabyrgalary: 4, 5
eseleu adisin butin parametrmen shaqyru
tortburysh qabyrgalary: 8, 10
eseleu adisin naqty parametrmen shaqyru
tortburysh qabyrgalary: 9.2, 11.5
eseleu adisin eki butin parametrmen shaqyru
tortburysh qabyrgalary: 8, 15
```

Мұнда **eseleu()** әдісі 3 рет асыра жүктеледі.

1) Бір бүтін типті параметрмен – 2;
2) Бір нақты типті параметрмен – 2.3;
3) Екі бүтін типті параметрмен – 2 және 3.

Әдісті асыра жүктеуде қайтарылатын мәннің типі ешқандай рөл атқармайды. Бірақ бір әдіске екі түрлі қайтару типін қолдану қате береді. Төменде **eseleu()** әдісіне екі түрлі (қайтарылатын мәннің типі бойынша) мән қайтару тәсілін қолдану мысалы көрсетілген.

```
// eseleu(int) әдісін бір рет жариялау жеткілікті
public Tortburysh eseleu(int k) {
  return new Tortburysh(a*k,b*k);
}
/* Қате! Әртүрлі мән қайтаратын болғанмен, eseleu(int) әдісін екінші рет жариялауға
болмайды. */
public void eseleu(int k) {
  Console.WriteLine("tortburysh parametrleri : "
          + a * k + ", " + b * k);
}
```

Комментарийден көрінгендей, **eseleu()** әдісінің екі нұсқасы екі түрлі мән қайтаратын болғанмен, ол асыра жүктеуге жеткіліксіз болып табылады.

Әдістерді асыра жүктеу программалау тілінің полиморфизм қасиетін сүйемелдейді, өйткені полиморфизмнің басты қағидасы: бір интерфейс – бірнеше әдіс мұнда жүзеге асырылады. Бұл түсініктілеу болуы үшін, тағы бір нақты мысал қарастырайық. Егер программалауда асыра жүктеу болмаса, әрбір әдістің өзіндік аты болуы тиіс. Бірақ программалауда мәліметтердің әртүрлі типтері үшін бір әдіс қажет болып жатады. Бізге санның абсолюттік мәнін анықтайтын функция қажет болсын делік. Асыра жүктеу сүйемелденбейтін тілде мұндай функцияның әртүрлі аты бар бірнеше түрін құру керек болар еді.

Мысалы, С тілінде **abs()** функциясы – бүтін сандардың абсолюттік мәні үшін, **labs()** функциясы – ұзын бүтін санның абсолюттік мәні үшін, **fabs()** – нақты санның абсолюттік мәні үшін болып кете береді. C# тілінде әдістер асыра жүктеледі, өйткені **System.Math** класында әртүрлі типтегі мәліметтерді өңдейтін бір ғана **Abs()** әдісі бар.

C# тілінде әдістің аты мен оның параметрлері тізімін белгілейтін *сигнатура* түсінігі анықталған. Асыра жүктеуге байланысты бұл түсінік бір класта сигнатуралары бірдей екі әдіс болмауы тиіс дегенді білдіреді. Сигнатура түсінігіне әдістен қайтарылатын мән типі кірмейді, өйткені компилятор асыра жүктеу жайлы шешім қабылдағанда, қайтарылатын мән типі есепке алынбайды. Сигнатураға **params** модификаторы да енгізілмеген.

Конструкторларды асыра жүктеу

Әдістер секілді конструкторлар да асыра жүктеледі. Ол объектілерді әртүрлі жолдармен құру мүмкіндігін береді. Мысал қарастырайық.

Мысал 2. Конструкторды асыра жүктеу.

```
using System;
class Tortburysh {
  public double a;
  public double b;
  public Tortburysh() {
    Console.WriteLine("Tortburysh() konstruktorynda");
    a=b=1;
  }
  public Tortburysh(double a) {
    Console.WriteLine("Tortburysh(double) konstruktorynda");
    this.a=a; this.b = a;
  }
  public Tortburysh(double a, double b) {
    Console.WriteLine("Tortburysh(double,double)
                       konstruktorynda");
    this.a=a; this.b = b;
  }
  // Көшіру конструкторы
  public Tortburysh(Tortburysh T) {
    Console.WriteLine("Tortburysh(Tortburysh)
                       konstruktorynda");
    this.a=T.a; this.b = T.b;
  }

  public void Shygaru() {
    Console.WriteLine("tortburysh qabyrgalary: " +a+ ", "+b);
  }
}
class Program  {
```

```
  static void Main() {
    Tortburysh ob1 = new Tortburysh();
    ob1.Shygaru();
    Tortburysh ob2 = new Tortburysh(2.3);
    ob2.Shygaru();
    Tortburysh ob3 = new Tortburysh(2.3, 4.5);
    ob3.Shygaru();
    Tortburysh ob4 = new Tortburysh(ob3);
    ob4.Shygaru();
  }
}
```

Бұл программаның нәтижесі:

```
Tortburysh() konstruktorynda
tortburysh qabyrgalary: 1, 1
Tortburysh(double) konstruktorynda
tortburysh qabyrgalary: 2.3, 2.3
Tortburysh(double, double) konstruktorynda
tortburysh qabyrgalary: 2.3, 4.5
Tortburysh(Tortburysh) konstruktorynda
tortburysh qabyrgalary: 2.3, 4.5
```

Бұл мысалда **Tortburysh()** конструкторы 4 рет асыра жүктеледі, әр орындалған сайын ол объектіні басқаша құрады. Қажетті конструктор new операторында көрсетілген аргументтерге байланысты шақырылады. Аргумент ретінде класс объектісін қабылдайтын конструктор көшіру конструкторы деп аталады. Осылай кластың конструкторын асыра жүктеу қосымша мүмкіндіктер береді екен.

Статикалық кластар және кластың статикалық мүшелері.

Кейде бір кластың басқа объектілерінен тәуелсіз түрде қолданылатын сол класс мүшесін анықтау керек болып жатады. Көбінесе класс мүшесіне қол жеткізу осы класс объектісі арқылы ұйымдастырылады, бірақ сонымен қатар объектінің нақты экземплярына сілтеме жаса-май-ақ, жеке қолданыла беретін класс мүшесін құруға болады.

Осындай мүше жасау үшін оны жариялаудың ең басында static түйінді сөзін көрсету жеткілікті. Егер класс мүшесі static болып жарияланса, онда басқа ешбір объектіге сілтеме жасамай-ақ, ол өз класының кез келген объектісін жасау үшін қолданыла алады.

Static түйінді сөзі арқылы айнымалыларды да және сонымен қатар әдістерді де жариялауға болады. static типіндегі мүшенің қалыпты мысалы ретінде операциялық жүйе арқылы шақырылып, программаның ең басында жарияланатын Main() әдісін есептеуге болады. static типіндегі мүшені кластан тыс қолдану үшін, осы класс атын нүкте-оператормен көрсету жеткілікті. Бірақ бұл үшін объект құру қажет емес.

Негізінде, static типіндегі мүшеге қол жеткізу объектіге сілтеме жасау жолымен емес, өз класының аты бойынша жүзеге асады. Сонымен, егер Timer класының мүшесі болып табылатын static типіндегі Count айнымалысына 10 мәнін беру керек болса, онда келесі код жолын қолдануға болады.

```
Timer.Count = 10;
```

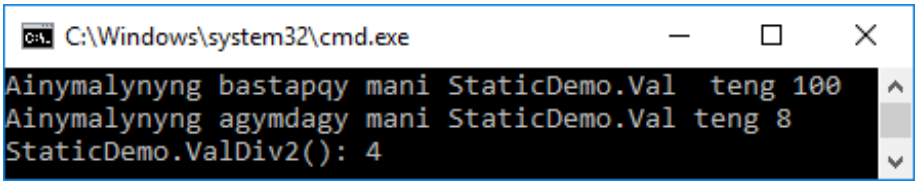
Бұл жазу формасы объект арқылы қарапайым экземпляр айнымалысын пайдалану кезіндегі сияқты болып жазылады, бірақ онда объектінің аты емес, класс аты көрсетіледі. Осындай тәсіл арқылы класс аты мен нүкте-операторды пайдаланып, static типіндегі әдісті де шақыруға болады.

static типінде жарияланған айнымалылар, негізінде, глобальді айнымалылар болып табылады. Объектілер өз класында жарияланған кезде, static типіндегі айнымалылардың көшірмесі жасалмайды. Оның орнына кластың барлық экземплярлары static типіндегі бір

айнымалымен бірге қолданылады. Мұндай айнымалы оны класта пайдалану алдында инициалданады.
Айнымалының инициализаторы тікелей көрсетілмеген жағдайда, ол сандық типте болса, нөл мәнімен, сілтемелік типте болса, бос мәнмен инициалданады, ал егер ол bool типінде болса, оған false мәні беріледі.
Сонымен, static типіндегі айнымалы әрқашанда белгілі бір мәнге ие болуы тиіс. static типіндегі әдістің қарапайым әдістен айырмашылығы оны бұл кластың объектісі экземплярын жасамай-ақ, өз класының атымен шақыруға болады. Мұндай шақыру мысалы бұрын келтірілген болатын. Ол C# кластары кітапханасындағы System.Math класына жататын static типіндегі Sqrt() әдісі болатын.
Төменде static типіндегі айнымалы және әдіс жарияланған программа мысалы көрсетілген.

```
// static модификаторын пайдалану.
using System;
class StaticDemo {
    // static типіндегі айнымалы.
    public static int Val = 100;
    // static типіндегі әдіс.
    public static int ValDiv2() { return Val/2;  }
        }

        class SDemo {
    static void Main() {
            Console.WriteLine("Ainymalynyng bastapqy mani " +
                                "StaticDemo.Val  teng " + StaticDemo.Val);

            StaticDemo.Val = 8;
            Console.WriteLine("Ainymalynyng agymdagy mani " +
            "StaticDemo.Val teng " + StaticDemo.Val);
            Console.WriteLine("StaticDemo.ValDiv2(): " + StaticDemo.ValDiv2());
    }
}
```

Бұл программа нәтижесі:



Мұнан көретініміз static типіндегі айнымалы оның класын-дағы кез келген объект құрылғанша, инициалданады.
static типіндегі әдістерді қолдануға бірсыпыра шектеулер қойылады:
• static типіндегі әдістерде this сілтемесі болмауы тиіс, өйткені мұндай әдіс белілі бір объектіге қатысты орындалмайды.
• static типіндегі әдісте осы кластың экземпляры әдісін емес, тек басқа static типіндегі әдістерді тікелей шақыруға рұқсат етілген. Мұның себебі – экземпляр әдістері нақты объектілермен жұмыс істейді, ал static типіндегі әдіс объект үшін шақырылмайды. Сондықтан, мұндай әдістің жұмыс істейтін объектілері болмайды.
• Дәл осындай шектеулер static типіндегі мәліметтерге де қойылады. static типіндегі әдіс үшін оның класында анықталған басқа static типіндегі мәліметтер тікелей қол жетімді болып табылады.
Ол, мысалы, өз класының экземпляры айнымалысымен жұмыс істей алмайды, өйткені онда оның өңдей алатын объектілері болмайды. Төмендегі мысалда static типіндегі ValDivDenom() әдісі жұмыс істей алмайды.

```
class StaticError {
```

```
    public int Denom =3;                // экземплярдың қалыпты айнымалысы
     public static int Val = 1024;      // статикалық айнымалы

    /* Қате! Статикалық әдісте статикалық емес айнымалыны тікелей қолданбайды. */
       static int ValDivDenom() {
           return Val/Denom;            // компиляциядан өтпейді!
    }
}
```

Бұл мысалда Denom жай қалыптағы айнымалы болып табылады, оны static типіндегі әдісте қолдана алмаймыз. Бірақ мұнда Val айнымалысын пайдалана аламыз, өйткені ол static болып жарияланған.
Осындай қателік төмендегі мысалда да бір класс ішіндегі статикалық әдістен статикалық емес әдісті шақыру кезінде де туындайды.

```
using System;
class AnotherStaticError {

     // Статикалық емес әдіс.
     void NonStaticMeth() {
          Console.WriteLine("NonStaticMeth() adisinde.");
     }

     /* Қате! Статикалық әдістен статикалық емес әдісті тікелей шақыруға болмайды. */
     static void staticMeth() {
          NonStaticMeth(); // компиляциядан өтпейді!
    }
}
```

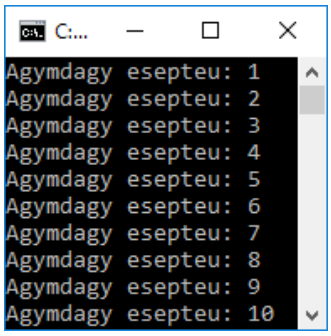
Мұнда компиляция кезінде статикалық әдістен статика-лық емес әдісті (экземпляр әдісін) шақыру тәсілі қате береді.
static типіндегі әдістен экземпляр әдістерін шақырып, оның класындағы экземпляр айнымалыларын сол класс объектілеріндегі тәрізді қолдануға болмайды.
Мұның себебі – нақты объект көрсетілмесе, экземпляр айнымалысын немесе әдісін қолдануға болмайды. Мысалы, төмендегі код фрагменті дұрыс жазылған болып саналады.

```
    class MyClass {

// Статикалық емес әдіс.
void NonStaticMeth() {
    Console.WriteLine("NonStaticMeth() adisinde. ");
}

       /* Объектіге сілтеме жасау арқылы статикалық әдістен  статикалық емес әдіс
          шақырыла алады. */
public static void staticMeth(MyClass ob) {
    ob.NonStaticMeth();        // бәрі дұрыс!
}
    }
```

Бұл мысалда NonStaticMeth() әдісі staticMeth() әдісінен MyClass типіндегі ob объектісіне сілтеме бойынша шақырылады.
static типіндегі өрістер нақты объектіге тәуелді болмайды, сондықтан олар бүкіл осы класта қолданылатын ақпарат сақтауға ыңғайлы.

Төменде осындай жағдайды көрсететін программа мысалы келтірілген. Бұл программада static типіндегі өріс бұрыннан бар объектілердің санын сақтау үшін қолданылады.

```
// static типіндегі өрісті  бұрыннан бар объектілер
// экземплярларын санау үшін пайдалану.
using System;
public CountInst() {  count++;  }
class CountInst {
    static int Count = 0;

     // Объект құру кезіндегі санауышты инкременттеу.
     public CountInst() {  count++;  }

// Объектіні жою кезінде санауышты декременттеу.
    ~CountInst() {  count--;  }
    public static int GetCount() { return count;  }
}

class CountDemo {
    static void Main() {
      CountInst ob;
      for(int i=0; i < 10; i++) {
         ob = new CountInst();
         Console.WriteLine("Agymdagy esepteu: " +
                                    CountInst.GetCount());
     }
    }
  }
```



Бұл программа нәтижесі:

```
Agymdagy esepteu: 1
Agymdagy esepteu: 2
Agymdagy esepteu: 3
…
Agymdagy esepteu: 10
```

Countlnst типіндегі объект құрылған сайын static типіндегі Count өрісі инкременттеледі, бірақ осындай объектіні жою кезінде Count өрісі декременттеледі. Сол себепті Count өрісі осы сәттегі объектілер санын сақтап тұрады. Ал бұл static типіндегі өрісті қолдану арқылы жүзеге асатын болады. Осындай санау тәсілін экземпляр айнымалысы арқылы ұйымдастыруға болмайды, өйткені ол бұл кластағы объектінің нақты экземплярын емес, бүкіл класты қамтиды.

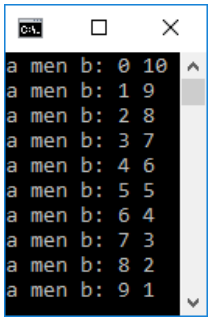
Төменде кластың статикалық мүшелерін қолданудың тағы бір мысалы келтірілген. Бұдан бұрынырақ кластар фабрикасы арқылы объектілерді құруға болатыны айтылған болатын. Ол мысалда фабрика статикалық емес әдіс болып жасалған еді, сондықтан ол фабрикалық әдіс тек алдын ала құрылған объектіге сілтеме жасау арқылы шақырылатын болаын.

Бірақ класс фабрикасын static типіндегі әдіс ретінде жүзеге асыру дұрысырақ болады, өйткені ол осы фабрикалық әдісті керекті объект құрмай-ақ шақыру мүмкіндігін беретін еді. Міне осы тәсіл келесі класс фабрикасын жүзеге асыратын мысалда көрсетілген.

```
// Статикалық класс фабрикасын қолдану.
using System;
class MyClass {
   int a, b;                    // MyClass класы үшін фабрика құру.
   static public MyClass Factory(int i, int j) {
```

```
        MyClass t = new MyClass();

        t.a = i;
        t.b = j;

        return t;   // объектіні қайтару
    }

    public void Show() { Console.WriteLine("a men b: " + a + " " + b); }
}

class MakeObjects {
      static void Main() {
          int i, j;
          // Фабриканы қолданып, объектілерді қалыптастыру.
          for(i=0, j = 10; i < 10; i++, j--) {
                MyClass ob = MyClass.Factory(i, j);    // объект  жасау
                ob.Show();
          }
          Console.WriteLine();
      }
}
```

Программа нәтижесі:



Программаның бұл нұсқасында фабрикалық Factory() әдісі оның класының аты арқылы былай шақырылады:

```
MyClass ob = MyClass.Factory(i, j);      // объект  жасау
```

Енді бұл кластың фабрикасын қолдану алдында MyClass класы объектісін құру қажет емес.

Статикалық конструкторлар

Конструкторды да static түрінде жариялауға болады. Көбінесе статикалық конструктор кластағы объектінің жеке экземпляры үшін емес, бүкіл класс үшін қолданылатын компоненттерді инициалдау үшін пайдаланылады.

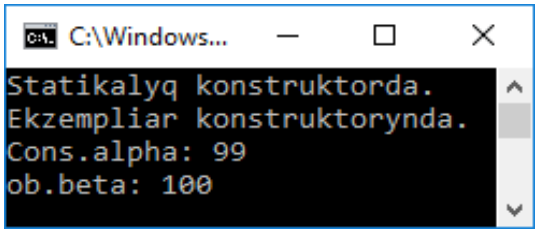
Сондықтан класс мүшелері осы кластың кез келген объектілерін құрудан бұрын статикалық конструктор арқылы инициалданады.

Төменде статикалық конструкторды пайдаланудың қарапайым мысалы келтірілген.

```
// Статикалық конструкторды қолдану.
using System;
class Cons {
    public static int alpha;
    public int beta;

    // Статикалық конструктор.
    static Cons() {  alpha = 99;
        Console.WriteLine("Statikalyq konStruktorda.");
     }

     // экземпляр конструкторы.
public Cons() {
```

```
beta = 100;  Console.WriteLine("Ekzempliar konStruktorynda.");
      class ConsDemo  {
          static void Main()  {
              Cons ob = new Cons();
              Console.WriteLine("Cons.alpha: " + Cons.alpha);
              Console.WriteLine("ob.beta: " + ob.beta);
          }
     }
}
```

Бұл прогамма нәтижесі мынадай болады:



Мұнда static типіндегі конструктор класс алғашқы рет жүктелгенде, экземпляр конструкторына дейін автоматты түрде шақырылатынына назар салыңыздар.

Бұдан шығатын жалпы қорытынды: статикалық конструктор кез келген экземпляр конструкторына дейін орындалуы тиіс. Оған қоса, статикалық конструкторларда қол жеткізу модификаторлары болмайды – оларға келісім бойынша қол жеткізіледі, сондықтан оларды программадан шақыруға болмайды.