2 дәріс. Конструкторлар және деструкторлар. Класс мүшелеріне қол жеткізуді басқару. Қасиеттер. Индексаторлар.

Дәрістің мақсаты: студенттерде конструкторлар, деструкторлар, қасиеттер және индексаторлардың қызметі және оларды пайдалану ерекшеліктері туралы түсініктерін көрсетуге қабілет қалыптастыру.

Осы дәрісті меңгеру нәтижесінде студенттер келесі қабілеттерге ие болады:

- конструкторлар, деструкторлар, қасиеттер және индексаторлардың қызметі туралы түсініктерін көрсету;
- конструкторлар, деструкторлар, қасиеттер және индексаторларды жариялау форматы бойынша білімдерін көрсету.

Конструкторлар

Жоғарыдағы программа мысалдарында **Shenber** типіндегі әрбір объектідегі экземпляр айнымалыларын қолдап келесідей операторлар тізбегі арқылы инициалдауға тура келген еді.

```
shenber1.xcoord = 2;
shenber1.ycoord =2;
shenber1.radius = 3;
```

Мұндай тәсіл әдетте C# тілінде тілінде кәсіби түрде жазылған кодтарда қолданылмайды. Оның үстіне, мұнда қате де кетуі мүмкін (сіз өрістердің бірін инициалдауды ұмытып та кетуіңіз де мүмкін ғой). Дегенмен, осындай сәттерде көмектесетін жақсы бір тәсіл бар, ол: конструкторларды қолдану.

Конструктор объектіні құру кезінде бірден инициалдайды. Конструктордың аты оның класс атымен бірдей болып келеді де, синтаксис жағынан ол әдіске ұқсайды. Бірақ конструкторлардың тікелей көрсетіліп қайтарылатын типі жоқ.

Төменде конструктордың жапы формасы келтірілген.

```
қатынасу_типі класс_аты(параметрлер_тізімі) {
   // конструктор тұлғасы
}
```

Көбінесе конструктор класта анықталған экземпляр айнымалыларының бастапқы мәндерін беру үшін қолданылады немесе басқа да толық қалыптасқан объект жасау үшін қажет етілетін орнату процедураларын орындау үшін пайдаланылады. Одан қоса, конструкторлар көбінесе класта шақырылатындықтан, оның қатынасу типінің модификаторы **public** болып келеді. Ал *параметрлер_тізімі* кейде бос болады, ал көбінесе көрсетілген бір немесе бірнеше параметрлер тізімінен тұрады.

Программада анықтасаңыз да немесе анықтамасаңыз да, бәрі бір барлық кластарда конструкторлар болады, өйткені C# тілінде, алдын ала келісім бойынша, конструктор автоматты түрде құрылады, ол барлық экземпляр айнымалыларын берілетін мәндермен инициалдайды. Мәліметтердің көптеген типтері үшін алдын ала келісім бойынша берілетін мән нөл болады, **bool** типі үшін – **false** мәні алынса, ал сілтемелі типтер үшін – бос мән алынады. Бірақ егер сіз программада өз конструкторыңызды анықтайтын болсаңыз, онда келісім бойынша құрылатын конструктор пайдаланылмайды.

Мысал 7.3. Конструкторды қолданудың қарапайым мысалы.

```
using System;
class Shenber {
   public int xcoord;
   public int ycoord;
```

```
   public double radius;
   public Shenber()
   {
      xcoord = 2;
      ycoord = 2;
      radius = 1;
   }
}
class Program {
   static void Main() {
      Shenber shenber1 = new Shenber();
      Console.WriteLine("shenber centri: " + shenber1.xcoord + ", " +
shenber1.ycoord + ", shenber radiusy – " + shenber1.radius);
      Console.ReadKey();
   }
}
```

Бұл мысалдағы **Shenber** класының конструкторы мынадай түрде берілген.

```
public Shenber()
{
   xcoord = 2;
   ycoord = 2;
   radius = 1;
}
```

Мұндағы конструктордың **public** түрінде белгіленгеніне назар аударыңыз. Ол өз класы сыртындағы кодтардан да шақырылатын болуы тиіс. Ол объектіні құру кезінде **new** операторында шақырылады. Мысалы, келесі жолда:

```
Shenber shenber1 = new Shenber();
```

Конструкциялаудан кейін **shenber1** объектісінің айнымалылары сәйкесінше 2, 2, 3 мәндерін алады. Сонымен, жоғарыда келтірілген кодты орындау нәтижесі мынадай болады.

```
shenber centri: 2, 2, shenber radiusy – 1;
```

Алдыңғы мысалда параметрсіз конструктор пайдаланған болатын. Кейбір жағдайларда ол жеткілікті болғанмен, көбінесе конструктор бір немесе бірнеше параметрдерді қабылдауы тиіс. Конструкторға параметрлер әдістерге енгізілетін сияқты түрде енгізіледі. Ол үшін параметрлерді конструктор атынан кейін жай жақшалар ішінде жариялау жеткілікті.

Мысал 7.4. Параметрленген **Shenber** конструкторын қолдану мысалы.

```
using System;
class Shenber {
   public int xcoord;
   public int ycoord;
   public double radius;
   public Shenber(int x, int y, int r)
   {
      xcoord = x;
      ycoord = y;
      radius = r;
   }
}
class Program {
   static void Main()
   {
      Shenber shenber1 = new Shenber(2,2,1);
      Console.WriteLine("shenber centri: " + shenber1.xcoord + ", " +
shenber1.ycoord + ", shenber radiusy – " + shenber1.radius);
       Console.ReadKey();
   }
```

```
}
```

Бұл кодты орындау нәтижесі мынадай болады:

```
shenber centri: 2, 2, shenber radiusy – 1;
```

Деструкторлар

Жоғарыда көрсетілгендей, new операторын қолдану кезінде жасалатын объектілер үшін компьютер жедел жадынан алынған бос орын көлемі (буфердегі) динамикалық түрде бөлінеді. Әрине, жедел жады көлемі шексіз емес, ол ерте ме, кеш пе, әйтеуір бітеді. Сондай кездерде, керекті объект құру үшін бос жады мөлшерінің болмай қалуы **new** операторының дұрыс орындалмауына алып келеді. Міне, сондықтан да компьютер жадын динамикалық түрде бөлудің кез келген схемасының басты функциясы пайдаланылмайтын объектілерден жадыны босатып, оны келесі жұмыстарға қолдануға дайындау болып табылады.

C# тілінде "*қоқысты жинау*" жүйесінің өзі компьютер жадын пайдаланылып болған объектілерден байқаусыз және программалаушының қатысуынсыз автоматты түрде босатады. "Қоқысты жинау" былай жүргізіледі. Егер объектіге сілтеме болмаса, онда ондай объект керексіз деп есептеледі де, ол орналасқан жады босатылып, буферге қосылады. Осылай "тазартылған" жады басқа объектілерге бөлініп берілетін болады. "Қоқысты жинау" программаны орындау барысында, анда-санда жүргізіліп отырады. Ол жұмыстан босаған бір-екі объект үшін орындала салмайды, өз кезегін күтеді. Сонымен, "қоқысты жинау" қай кезде болатынын алдын ала білуге немесе болжауға болмайды.

C# тілінде "қоқысты жинау" жүйесі арқылы қажет болмайтын объектіні жою алдында орындалатын әдісті шақыруды анықтау мүмкіндігі бар. Мұндай әдіс деструктор деп аталады және ол объектімен жұмыс істеу мерзімін аяқтауға кепілдік беретін ерекше жағдайларда пайдаланыла алады. Мысалы, деструктор босатылуға тиіс объект алып тұрған жүйелік ресурсты кепілді түрде босату үшін қолданылады. Бірақ деструкторлар тек сирек туындайтын ерекше жағдайларда ғана қолданылатын құралдар болып табылатынын айта кету керек.

Төменде деструктордың жалпы жазылу формасы келтірілген:

```
~класс_аты() {
   // деструктор коды
}
```

мұндағы *класс_аты* – нақты бір кластың аты. Деструктор конструктор тәрізді жарияланады, тек оның аты алдына "тильда" (**~**) белгісі қойылады. Деструктордың қайтарылатын типі мен оған берілетін аргументтері болмайды.

Деструкторды бір класқа қосу үшін, оны класс құрамына мүше етіп енгізу жеткілікті. Ол өз класындағы белгілі бір объектіні жою керек болған сайын іске қосылады. Деструкторда объектіні жою алдында орындалуға тиіс әрекеттерді көрсетуге болады.

Мысал 7.5. Төменде деструкторды қолдануды көрсететін программа мысалы келтірілген. Ол программада көптеген объектілер құрылады және өшіріледі. Ол процестің орындалуы барысындағы кейбір сәттерде "қоқысты жинау" іске қосылып, қажетсіз объектілерді өшіру үшін деструкторлар шақырылып жатады.

```
using System;
class Destruct {
   public int x;
   public Destruct(int i) {
      x = i;
   }
   // Объектіні өшіру кезінде шақырылады
   ~Destruct() {
      Console.WriteLine("Joiylady " + x);
   }
   // Объектіні жасайды да және оны бірден жояды
```

```
   public void Generator(int i) {
      Destruct o = new Destruct (i);
   }
}
class Program {
   static void Main() {
       int count;
       Destruct ob = new Destruct(0);
       /* Ал енді көптеген объектілер жасалады.
Белгілі бір сәттерде "қоқыс жинау" орын алады.
Ескерту: "қоқыс жинауды" екпінді ету үшін жасалатын объектілер
санын көбейту керек шығар */
       for (count=1; count < 100000; count++)
       ob.Generator(count);
       Console.WriteLine("Dayin!");
   }
}
```

```
...
Joiylady 4403
Joiylady 4402
Joiylady 4401
Joiylady 4400
Joiylady 4399
Joiylady 4398
Joiylady 4397
Joiylady 4396
Joiylady 4395
Joiylady 4394
Joiylady 4393
...
```

Программа орындалуының нәтижесі:

Бұл программа былай жұмыс істейді. Конструктор **x** айнымалысын белгілі бір мәнмен инициалдайды. Бұл мысалда **x** айнымалысы объект идентификаторы рөлін атқарады. Ал деструктор объект жұмысын бітірген соң, **x** айнымалысының мәнін шығарады. **Destruct** типіндегі объектіні жасап және бірден жоятын **Generator()** әдісі ерекше қызығушылық туғызады. Алдымен **Program** класында **Destruct** типіндегі **ob** бастапқы объектісі жасалады, ал сонан соң біртіндеп кезекпен 100 мың объект жасалып өшіріледі. Осы процестің әртүрлі сәттерінде "қоқысты жинау" орын алады. Оның қаншалықты жиі орындалуы – бірнеше факторларға байланысты болады, оның ішінде бастапқы бос жады көлемі, пайдаланылатын операциялық жүйе типі және т.с.с. бар. Дегенмен белгілі бір сәтте деструктор арқылы қалыптасатын хабарламалар пайда бола бастайды. Егер де олар программа жұмысының соңына дейін, яғни "Дайын!" деген хабарлама шыққанша пайда болмаса, онда жасалынатын объектілердің шектеулі санын **for** циклiндегі қадамдар санын көбейте отырып арттыру керек.

Тағы бір маңызды ескерту: **~Destruct()** деструкторында **WriteLine()** әдісі тек осы мысалдың көрнекті болып көрінуі үшін ғана шақырылады. Көбінесе деструктор тек өз класында анықталған экземпляр айнымалыларына ғана әсер етуі керек.

Деструкторларды шақыру реттілігі дәлме-дәл анықталмағандықтан, оларды программаның орындалуы барысындағы белгілі бір сәтте жүзеге асатын әрекеттерді орындау үшін қолдануға болмайды. Дегенмен, "қоқысты жинауды" қолдап инициалдау көптеген жағдайларда ұсынылмайды, өйткені бұл программаның тиімділігін төмендетуге әкеліп соқтыруы мүмкін. Оның үстіне, "қоқысты жинау" жүйесінің өз ерекшеліктері бар – егер тіпті "қоқысты жинауды" тікелей орындауды сұратсақ та, нақты объектінің қашан жойылатынын бәрі бір алдын ала білуге болмайды.

Мысал 7.6. Кесінді класын құрыңыз. Класс кесіндінің екі ұшының координаталарын анықтайтын 4 өрістен тұрады. Келесі әдістерді жүзеге асырыңыз:

1. кесіндінің ұзындығын анықтау;
2. өрістерді инициалдауға арналған конструкторлар құру.

```
using System;
class Kesindi {
   public int x1, y1, x2, y2;
   public Kesindi() { x1 = y1 = 0; x2 = y2 = 1; }
   public Kesindi(int a, int b, int c, int d) {
      x1 = a; y1 = b; x2 = c; y2 = d;
   }
   private double uzyndygy() { return Math.Sqrt
          (Math.Pow(x2 - x1, 2) + Math.Pow(y2 - y1, 2)); }
   public void Shygaru()
   {
      Console.Write("kesindi koordinatalary: ({0}, {1}) zhane
          ({2}, {3}), onyn uzyndygy - {4:#.##}", x1, y1, x2,
```

```
            y2, uzyndygy());
    }
}
class Program {
    static void Main() {
        Kesindi A = new Kesindi();
        Kesindi B = new Kesindi(2, 2, 7, 8);
        Console.Write("A ");
        A.Shygaru();
        Console.WriteLine();
        Console.Write("B ");
        B.Shygaru();
        Console.WriteLine();
        Console.ReadKey();
    }
}
```

Программаның орындалу нәтижесі:

```
A kesindi koordinatalary: (0, 0) zhane (1, 1), onyn uzyndygy –
1,41
B kesindi koordinatalary: (2, 2) zhane (7, 8), onyn uzyndygy –
7,81
```

Класс мүшелеріне қол жеткізу (қатынасу) модификаторлары

Класс мүшелеріне қол жеткізуді (қатынасуды) басқару жұмысы төрт түрлі *қатынасу модификаторы* арқылы: **public**, **private**, **protected** және **internal** ұйымдастырылады. **protected** модификаторы тек мұралауға байланысты іс-шараларды атқарады. Ал **internal** модификаторы негізінен кеңейтілетін программаларға немесе кітапханаларға байланысты жинақтау (*сборки)* мақсатында қолданылады.

Егер класс мүшесі **public** спецификаторымен белгіленсе, оны программадағы кез келген басқа кодтарда, мысалы, басқа класта анықталған әдістерде де қолдануға болады.

Ал егер класс мүшесі **private** спецификаторымен белгіленсе, онда оны осы кластың басқа мүшелерінде ғана пайдалануға болады. Сондықтан, басқа кластардың әдістері бұл кластың жабық мүшелерін (**private**) пайдалана алмайды. Ал егерде ешбір спецификатор көрсетілмесе, онда класс мүшесі, келісім бойынша, жабық болып саналады. Сол себепті кластың жабық мүшелерін құрған кезде **private** спецификаторын жазу міндетті емес.

Қатынасу спецификаторы жеке мүшенің типін сипаттау бөлігінің алдында көрсетіледі. Бұл класс мүшесін жариялау операторы осыдан басталуы тиіс дегенді білдіреді. Сонымен, программаларда қатынас түрлері былай көрсетіледі:

```
public  string  avtor;
private int zhyl;
private  bool  ishki_nukte(int x, int y) { // ...}
```

public және **private** модификаторларының айырмашылығын көрсету үшін келесі мысалды қарастырамыз.

Мысал 9.1. Класс мүшелерін қолданудағы **public** және **private** қатынасу түрлерінің (модификаторлары) айырмашылықтары.

```
using System;
class Shenber {
   private int xcoord;    // private арқылы жабық қатынас
   int ycoord;            // келісім бойынша жабық қатынас
   public int radius;     // public арқылы ашық қатынас
   // Класс мүшелері осы класс ішіндегі
   // жабық (private) мүшелерді пайдалана алады
```

```
   public void SetXcoord(int x) {
      xcoord = x;     }

   public int GetXcoord() {
      return xcoord;
   }        // әдіс

   public voidSetYcoord(int y) {
      ycoord = y;
   }

   public int GetYcoord()  {
      return ycoord;          // әдіс
   }
}

class Program {
   static void Main() {
      Shenber shenber1 = new Shenber();
      // Келесі қатынас түрлері кластың xcoord және ycoord
      // мүшелерін қолдануға рұқсат етпейді.
      // shenber1.xcoord = 10;   // Қате! xcoord – жабық мүше!
                                 // Әдіс керек
      // shenber1.ycoord = -10;  // Қате! ycoord – жабық мүше!
      // Осы кластың xcoord, ycoord мүшелерін пайдалану
      // оның өз класындағы әдістер арқылы ғана рұқсат етіледі.
      shenber1.SetXcoord(10);
      shenber1.SetYcoord(-10);
      // Бұл кластың radius мүшесін тікелей қолдана аламыз,
      // өйткені ол (public) ашық мүше ретінде жарияланған.
      shenber1.radius = 2;
      Console.WriteLine("shenber    centrinin    x    coordinatasy    =    " +
shenber1.GetXcoord ());  // жабық x координатасы
      Console.WriteLine("shenber    centrinin    y    coordinatasy    =    " +
shenber1.GetYcoord ());      // жабық y координатасы
      Console.WriteLine("shenber radiusy = " +
                        shenber1.radius);    // ашық радиус
   }
}
```

Программа нәтижесі:

```
shenber centrinin x coordinatasy = 10
shenber centrinin y coordinatasy = -10
shenber radiusy = 2
```

Мұнда **Shenber** класында **xcoord** мүшесі **private** сөзі арқылы тікелей көрсетілген, **ycoord** мүшесі келісім бойынша **private** болып саналады, ал **radius** мүшесі **public** болып белгіленген. Сонымен, **xcoord** пен **ycoord** бұл кластан тысқары аймақтағы кодта пайдаланыла алмайды, өйткені олар жабық (**private**) түрде берілген (олар үшін ашық әдіс керек). Оларды **Program** класында тікелей қолдана алмаймыз, олар тек **SetXcoord()** және **GetXcoord()** тәрізді ашық (**public**) әдістер арқылы ғана пайдаланылады.

Ашық және жабық қатынасуды ұйымдастырудың жалпы қағидалары мыналар:

- Тек класс ішінде пайдаланылатын мүшелер жабық болып көрсетілуі тиіс.
- Анықталған мәндер шегінен асып кететін экземпляр мәліметтері жабық болуы керек, оларды пайдалану кезінде ашық әдістер арқылы шектен шықпағанын тексеріп отыру қажет.
- Егер мүшенің өзгеруі оның берілген шекті аймағынан асып кетуіне алып келсе, яғни объектінің басқа аспектілеріне әсер ететін болса, мұндаймүше жабық болуы керек, ал оны пайдалану бақыланатын болуы тиіс.

- Егер объектіге зиян келтіре алатын мүшелер дұрыс пайдаланылмайтын болса, олар жабық болуы тиіс. Ондай мүшелерді пайдалануды, қате болмайтындай етіп, ашық әдістер арқылы ұйымдастыру керек.
- Жабық мәліметтердің мәндерін тағайындайтын (орнататын) және қабылдайтын әдістер ашық болуы тиіс.
- Экземпляр айнымалыларының жабық болуына ешқандай негіз болмайтын жағдайда ғана олардыашық етіп алуға болады.