8-лекция. Интерфейстер. Контейнерлік кластар

Интерфейстерді сипаттау және қолдану.

Объектілерді клондау, салыстыру, реттеу үшін стандартты .NET интерфейстерін қолдану.

Контейнер (коллекция) түсінігі.

Стандартты .NET топтамаларын (коллекция) қолдану.

Интерфейстер

Интерфейс туралы жалпы мағлұматтар

- *Интерфейс* кластардың арнайы бір түрі, ол абстрактілі кластың «шеткі бір түрі (жағдайы)» болып табылады. Интерфейсте туынды класта жүзеге асырылуы тиіс абстрактілі тәсілдер, қасиеттер және индексаторлардың жиыны беріледі.
- Интерфейс оны іске асыратын кластар қолдайтын әрекет-терді анықтайды.
- Интерфейсті қолданудың негізгі идеясы – осындай класс объектілерін бірдей жолдармен қолдануға мүмкіндік беру.
- Әрбір класс интерфейс элементтерін өз қалауынша анықтай алады. Осылайша полиморфизмге қол жеткізіледі: әртүрлі класс объектілері бір тәсілдің шақыруына әртүрлі жауап әрекеттер орындай алады.
- Интерфейс синтаксисі класс синтаксисіне ұқсас:

[атрибуттар] [спецификаторлар] interface аты [: ата_тегі]

интерфейс_тұлғасы [;]

- Бір интерфейсте бірнеше интерфейс қасиеттері мұралану мүмкіндігі бар, мұндайда олардың ата тектері үтір арқылы бөліне отырып жазылады.
- *Интерфейс тұлғасын* абстрактілі тәсілдер, қасиеттер мен индексаторлар шаблондары , оқиғалар құрайды.
- Интерфейс құрамында тұрақтылар, өрістер, операциялар, конструкторлар, деструкторлар, типтер және кез келген статикалық элементтер болмауы тиіс.

```
interface IAction
{
    void Draw();
    int Attack(int a);
    void Die();
    int Power { get; }
}
```

Интерфейстердің қолданылу аймағы

- Егер қандай да бір әрекеттер жиынын қолдану тек осы әрекеттерді әртүрлі тәсілдермен орындайтын нақты бір кластар иерархиясы үшін маңызды болса, бұл жиынды иерархияның абстрактілі базалық класының виртуалды тәсілдері түрінде берген жөн.
- Иерархия шегінде (ішінде) бірдей жұмыс істейтіннің барлығын толығымен базалық класта анықтаған жөн.
- Интерфейстер көп жағдайда әртүрлі иерархия объектілерінің ортақ қасиеттерін көрсету үшін қолданылады.

Интерфейстің абстрактілі кластан айырмашылығы

- Интерфейс элементтерінің үнсіз келісім бойынша қабылданған қатынас құру спецификаторы public болып табылады және олардың нақты түрде көрсетілген спецификаторлары болуы мүмкін емес;
- Интерфейсте өрістер мен жай тәсілдер болуы мүмкін емес — интерфейстің барлық элементтері абстрактілі болуы тиіс;
- Ата тектер тізімінде интерфейсі көрсетілген класс оның барлық элементтерін анықтауы тиіс, ал абстрактілі кластың ұрпағы тегінің абстрактілі тәсілдерінің кейбір бөлігін қайта анықтамаса да болады (мұндайда туынды класс та абстрактілі болып табылады);
- Кластың өз ата тектері тізімінде бірнеше интерфейсі болуы мүмкін, мұндайда класс олардың барлық тәсілдерін анықтауы керек.

Интерфейстің жүзеге асырылуы

- C# тілі кластар үшін жалқы мұралауды және интерфейстер үшін көптік мұралауды сүйемелдейді.
- Интерфейстегі тәсілдердің сигнатуралары мен олардың жүзеге асырылуы бір-біріне толықтай сәйкес келуі тиіс.
- Интерфейстің орындалатын элементтері үшін класта public спецификаторын көрсету керек.
- Бұл элементтерге класс объектісі арқылы да немесе соған сәйкес интерфейс типінің объектісі арқылы да қатынас құруға болады.

Мысал

```
interface Iaction
  {  void Draw();  int Attack( int a );  void Die();  int Power { get; }  }
class Monster : IAction
   {  public void Draw() { Console.WriteLine( «Мұнда " + name + “болды”); }
      public int Attack( int ammo_ )  {
         ammo -= ammo_;
         if ( ammo > 0 ) Console.WriteLine( "Ба-бах!" ); else  ammo = 0;
         return ammo;
      }
      public void Die()
           { Console.WriteLine( "Monster " + name + " RIP" ); health = 0; }
      public int Power { get  { return ammo * health; }
    }
```

```
Monster Vasia = new Monster( 50, 50, "Вася" );  // Monster  класының
                                                //  объектісі
Vasia.Draw();                                   // Мұның нәтижесі: Вася болады
 IAction Actor = new Monster( 10, 10, "Маша" );  // интерфейс типті объект
Actor.Draw();                                   // нәтижесі: Мұнда Маша болды
```

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 орындалады. 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 интерфейстер 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 әрекеттерін тағайындайтын көптеген стандартты интерфейстер анықталған. Мысалы, **IComparable** интерфейсі объектілерді «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** интерфейстерін жүзеге асырады.
- Стандартты интерфейстерді сүйемелдейтін өз кластарымызды 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

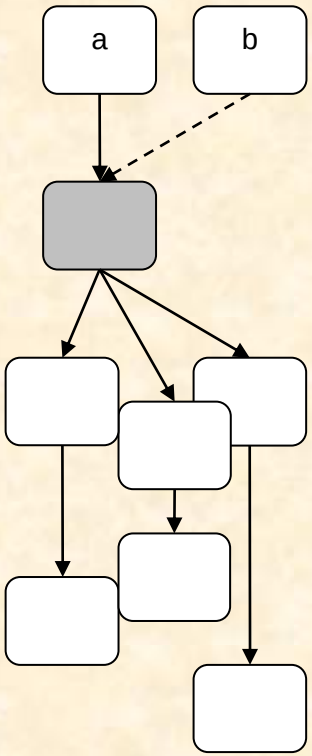
```
class Monster : IComparable
{ public int CompareTo( object obj )        // интерфейсті жүзеге асыру
      {  Monster temp = (Monster) obj;
         if ( this.health > temp.health ) return  1;
         if ( this.health < temp.health ) return -1;
         return 0;  }
  ... }
  class Class1
  {   static void Main()
      {  const int n = 3;
         Monster[] stado = new Monster[n];
         stado[0] = new Monster( 50, 50, "Вася" );
         stado[1] = new Monster( 80, 80, "Петя" );
         stado[2] = new Monster( 40, 10, "Маша" );
         Array.Sort( stado );              // реттеу мүмкін болды
}}
```

Параметрлері берілген интерфейстер

```
class Program {
    class Elem : IComparable<Elem>
    {  string data;
       int key;
       ...
       public  int  CompareTo( Elem obj )
       {   return key - obj.key;  }
    }
    static void Main(string[] args)
    {
       List<Elem> list = new List<Elem>();
       for ( int i = 0; i < 10; ++i ) list.Add( new Elem() );
       list.Sort();
       ...
    }
}
```

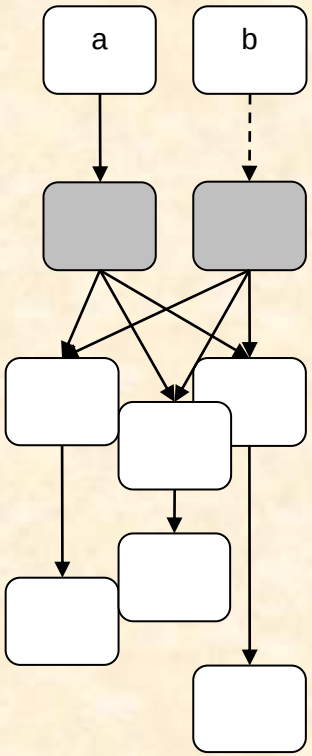
Объектілерді клондау

- *Клондау* — объект көшірмесін жасау. Объект көшірмесі клон деп аталады.

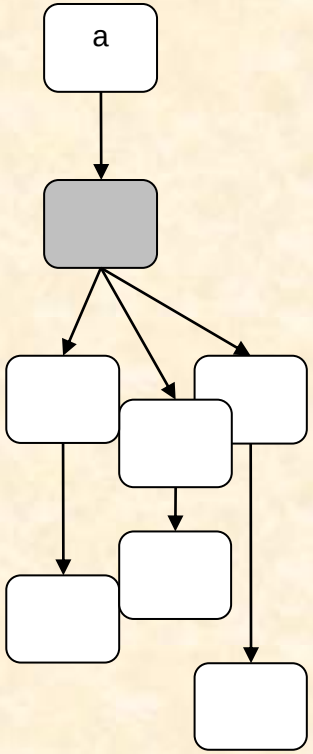


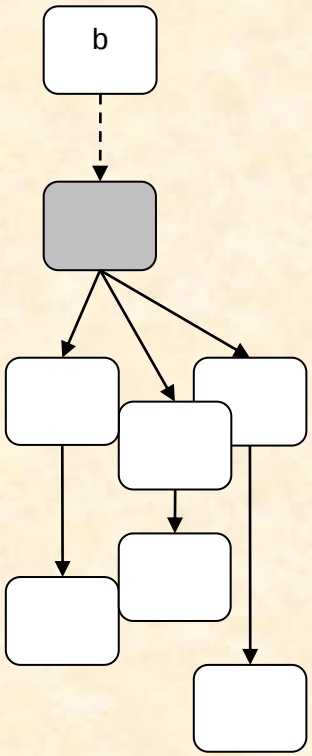
а) меншіктеу

b = a



б) үстіртін клондау





в) терең клондау

Клондау түрлері

- Бір сілтемелі типті объектіні екіншісіне меншіктеу кезінде объектінің өзі емес, сілтеме көшіріледі (*а* суреті).
- Егер жадының басқа аймағына объект өрісін көшіру қажет болса, онда объектінің object класынан мұраланатын MemberwiseClone тәсілін пайдалануға болады. Бұл кезде сілтемелер болып табылатын объект өрістері сілтеме жасап тұрған объектілер көшірілмейді (*б* суреті). Бұл *үстіртін клондау* деп аталады.
- Толықтай тәуелсіз объектілерді құру үшін *терең клондау* қажет етіледі, мұндайда компьютер жадында толығынан объектілер бұтағының дубликаты құрылады (*в* суреті).
- Терең клондау алгоритмі айтарлықтай күрделі болып келеді, себебі ол объектінің барлық сілтемелерін рекурсивті түрде қарастырып шығуды және циклдық тәуелділіктерді қадағалап отыруды талап етеді.
- Өзіндік клондау алгоритмдері бар объект **ICloneable** интерфейсінің мұрагері ретінде жариялануы және оның жалғыз **Clone** тәсілін қайта анықтауы тиіс.

Контейнерлік кластар

Абстрактілі мәліметтер құрылымдары

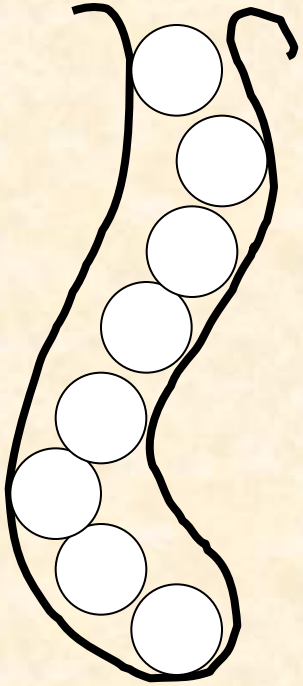
- *Жиым* — бір типтегі шамалардың шектелген жиыны. Жадының үзіліссіз біртұтас аймағында орналасады және элементтерге индекстері бойынша тікелей (кездейсоқ) қатынас құруға мүмкіндік береді. Жиымға жады онымен жұмыс істеу басталғанға дейін бөлінеді және кейіннен өзгермейді.
- Тізімде әрбір элемент келесісімен және, мүмкіндігінше, алдыңғысымен де байланысты болады. Тізімдегі элементтер саны программаның жұмысы барысында өзгеруі мүмкін. Тізімнің әрбір элементінде осы элементті идентификациялайтын (анықтайтын) *кілт* болады.
  - *бірбайланысты, екібайланысты*
  - *сақиналы*
- *Хеш-кесте (ассоциативті кжиым*, сөздік*)* — элементтеріне қатынас құру олардың нөмірі бойынша емес, кілті бойынша жүзеге асырылатын жиым (яғни, бұл «кілт-мән» жұптарынан тұратын кесте).

Стек

Стек — элементтерді қосу және одан элементтерді алу стек төбесі деп аталатын бір басынан ғана орындалатын бірбағытты тізімнің жеке жағдайы (түрі).

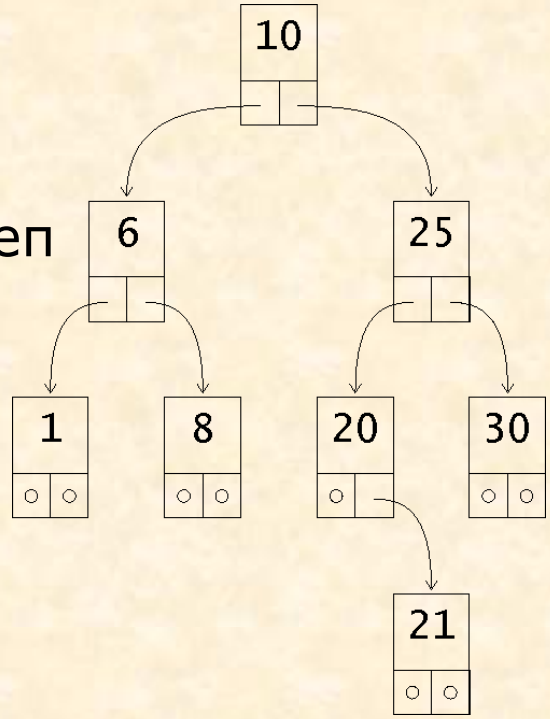
Стекпен орындалатын басқа операциялар анықталмаған.

Таңдап алу кезінде элемент стектен шығарылады.



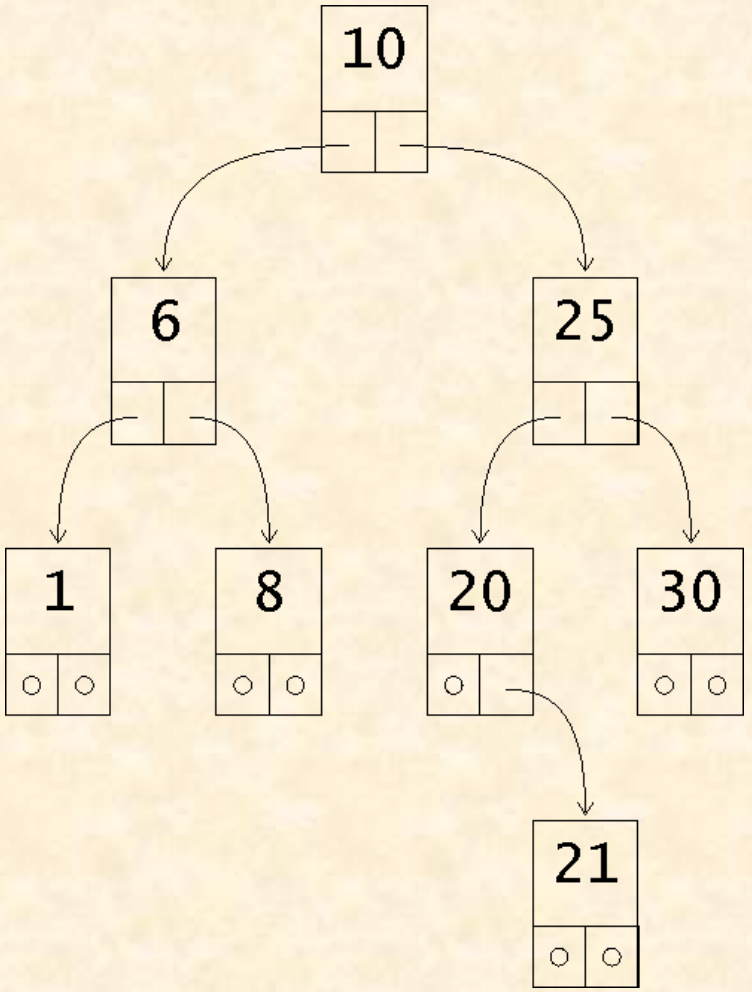
- *Кезек* — элементтерді қосу бір басынан, ал таңдау екінші басынан орындалатын бірбағытты тізімнің жеке жағдайы (түрі). Кезекпен орындалатын басқа операциялар анықталмаған. Таңдау кезінде элемент кезектен шығарылады.

- *Бинарлы бұтақ* — әрқайсысы мәліметтерден бөлек әртүрлі бинарлы ішкі бұтақтарға екіден артық емес сілтемесі бар түйіндерден тұратын динамикалық мәліметтер құрылымы.
- Әрбір түйінге дәл бір сілтеме болады. Бастапқы түйін *ағаш түбірі* деп аталады.
- Ішкі ағаштары болмайтын түйін *жапырақ* деп аталады. Шығатын түйіндер ата *текте*рі, ал кіретіндері *ұрпақтары* деп аталады.
- *Бұтақ биіктігі оның түйіндері орналасқан деңгейлер санымен анықталады.*



Іздеу бұтағы

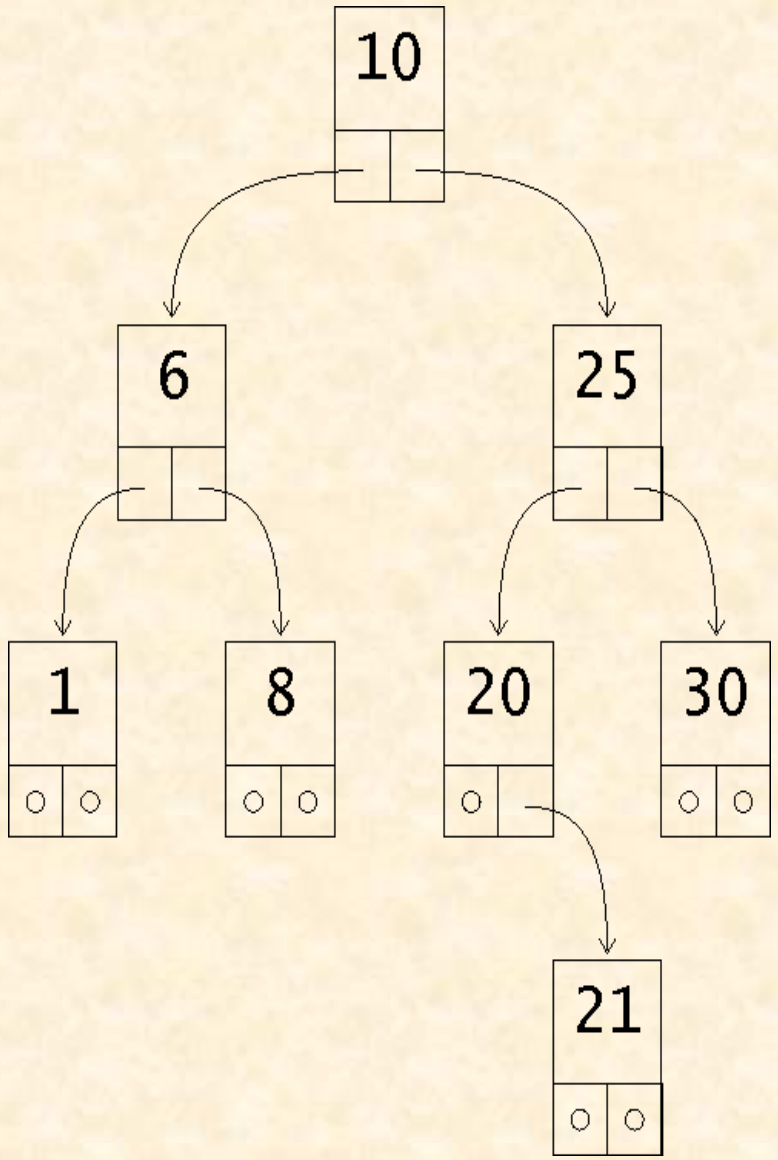
- Егер бұтақ әрбір түйіні үшін оның сол жақ ішкі бұтағының барлық кілттері осы түйін кілтінен кіші, ал оның оң жақ ішкі бұтағының барлық кілттері одан үлкен болатындай етіп ұйымдастырылса, ол *іздеу бұтағы* деп аталады. Бірдей кілттерге жол берілмейді. Іздеу бұтағында элементті түбірден бастап және әрбір түйіндегі кілт мәніне байланысты оң жақ немесе сол жақ ішкі бұтаққа өте отырып, кілті бойынша табуға болады.



Бұтақты қарап шығу (обход дерева)

```
procedure print_tree( бұтақ );
begin
    print_tree( сол_жақ_ішкі_бұтақ)
    түбірге бару
    print_tree( оң_жақ_ішкі_бұтақ )
end;
```

1 6 8 10 20 21 25 30



- *Граф* — әртүрлі түйіндерді байланыстыратын түйіндер мен қабырғалар жиыны. Көптеген өмірдегі нақты практикалық есептерді графтар терминдері арқылы сипаттауға болады, бұл мүмкіндік оларды программалар жазуда жиі қолданылатын мәліметтер құрылымына айналдырады.
- *Жиын* — реттелмеген элементтер жиыны. Жиындар үшін келесі операциялар анықталған:
  - элементтің жиынға тиістілігін тексеру
  - элементті қосу және шығару (жою)
  - жиындардың бірігуі, қиылысуы және айырмалары.
- Осы мәліметтер құрылымдарының барлығы *абстрактілі* деп аталады, себебі онда мүмкін операциялардың жүзеге асырылуы көрсетілмейді.

Контейнерлер

- *Контейнер (коллекция – топтама)* – абстрактілі мәліметтер құрылымын жүзеге асыратын стандартты класс.
- Топтаманың әрбір типі үшін сақталатын мәліметтердің нақты типінен тәуелсіз оның элементтерімен жұмыс істеу тәсілдері анықталған.
- Топтамаларды қолдану программаларды құру мерзімін қысқартуға және олардың сенімділігін арттыруға мүмкіндік береді.
- Топтаманың әрбір түрі өзінің мәліметтермен орындалатын операциялар жиынын сүйемелдейді және бұл операция-лардың орындалу жылдамдығы әртүрлі болуы мүмкін.
- Топтама түрін таңдау программада мәліметтермен қандай әрекеттер орындау қажеттігіне және оның орындалу жылдам-дығына қандай талаптар қойылатынына байланысты болады.
- .NET кітапханасында көптеген стандартты контейнерлер анықталған.
- Олар сипатталған негізгі атаулар кеңістігі — System.Collections, System.Collections.Specialized және System.Collections.Generic

System.Collections

ArrayList	Өз көлемін динамикалық түрде өзгертетін жиым
BitArray	Биттік мәндерді сақтауға арналған ықшамды жиым
Hashtable	Хэш-кесте
Queue	Кезек
SortedList	Кілттер бойынша реттелген топтама. Оның элементтерімен қатынас құру — кілт немесе индекс бойынша атқарылады
Stack	Стек

Параметрлері бар топтамалар (коллекциялар) (класс-прототиптер, generics)

- параметрлері ретінде мәліметтер типтері болатын кластар

Класс-прототип (2.0 нұсқада)	**Жай класс**
Dictionary<K,T>	HashTable
LinkedList<T>	—
List<T>	ArrayList
Queue<T>	Queue
SortedDictionary<K,T>	SortedList
Stack<T>	Stack

List класын пайдалану мысалы

```
using System;
using System.Collections.Generic;
namespace ConsoleApplication1{
class Program {
        static void Main()  {
            List<int> lint = new List<int>();
            lint.Add( 5 ); lint.Add( 1 ); lint.Add( 3 );
            lint.Sort();
            int a = lint[2];
            Console.WriteLine( a );
            foreach ( int x in lint ) Console.Write( x + " ");
}}}
```

Назарларыңыз

үшін рахмет!