

Paul Deitel dhe Harvey Deitel

Programimi i Orientuar në Objekte

(përmbledhje të kapitujve dhe ushtrime)

përzgjedhi dhe shqipëroi Ridvan Bunjaku

shkurt – maj 2023

Përmbajtja

4. Hyrje në klasa, objekte, metoda dhe stringje	7
4.1 Hyrje	7
4.2 Testimi i një klase <i>Llogari (Account)</i>	7
4.2.1 Instancimi i një Objekti – Fjala kyçe <i>new</i> dhe konstruktorët	8
4.2.2 Thirrja e metodës <i>GetName (MerreEmrin)</i> të klasës <i>Account (LlogariEBankes)</i>	9
4.2.3 Futja e një emri nga përdoruesi	9
4.2.4 Thirrja e metodës <i>Set</i> të klasës	9
4.3 Klasa me ndryshore të instancës dhe metoda <i>Set</i> dhe <i>Get</i>	9
4.3.1 Deklarimi i klasës	10
4.3.2 Fjala kyçe <i>class</i> dhe trupi i klasës	10
4.3.3 Variabla (ndryshorja) e instancës e tipit <i>string</i>	11
4.3.4 Metoda <i>SetName</i>	11
4.3.5 Metoda <i>GetName</i>	12
4.3.6 Modifikuesit (ndryshuesit) e qasjes <i>private</i> dhe <i>public</i>	12
4.3.6 Diagrami i klasës në UML (Unified Modeling Language – Gjuha e unifikuar e modelimit)	13
4.4 Krijimi, kompajlimi dhe ekzekutimi i një projekti me dy klasa	14
4.5-6 Inxhinierimi i softuerit me metodat <i>Set</i> dhe <i>Get</i> , <i>Property</i>	14
4.6.2 Klasa me variabël të instancës dhe një <i>property</i>	16
4.6.3 Diagrami UML i klasës me një <i>property</i>	17
4.7 <i>Property</i> -t e implementuara automatikisht (auto-implemented)	17
4.8 Inicializimi i objekteve me konstruktorë	18
4.8.1 Deklarimi i konstruktorit për inicializim specifik të objektit	19
4.8.2 Inicializimi i objekteve kur krijohen	19
4.9 Klasa me variabël <i>decimal</i> të instancës	21
4.9.2 Klasa që krijon dhe përdor objekte	23
7. Metodat: vështrim më i thellë	25
7.1 Hyrje	25
7.2 Paketimi i kodit në C#	25
7.2.1 Modularizimi i programeve	25
7.3 Metodat <i>static</i> -e, variablat <i>static</i> -e dhe klasa <i>Math</i>	26
7.4 Metodat me shumë parametra	28
7.5 Informata detale për deklarimin dhe përdorimin e metodave	30

7.6 Shndërrimi i tipeve të argumenteve	31
7.7 .NET Framework Class Library (Libraria e klasave e .NET Framework)	32
7.8 Studim i rastit: Gjenerimi (prodhimi) i numrave të rastësishëm	33
7.9 Studim i rastit: Një lojë e fatit; Hyrje në Enumeracione (Enumerations)	36
7.10 Skopi (scope, shtrirja, fushëveprimi) i deklarimeve	39
7.11 Steku i thirrjes së metodave (Method-Call Stack) dhe regjistrat e aktivizimit (activation records)	41
7.12 Mbingarkimi i metodës (method overloading)	45
7.13 Parametrat opsionalë	46
7.14 Parametrat e emërtuar (named parameters)	47
7.16 Rekursioni	48
7.16.1 Rastet bazë dhe thirrjet rekursive	48
7.16.2 Llogaritjet rekursive të faktorielit	49
7.17 Tipet vlerë vs Tipet referencë	50
7.18 Pasimi (dërgimi) i argumenteve me vlerë (by value) dhe me referencë (by ref.)	51
7.18.1 Parametrat ref dhe out	51
8. Vargjet (Arrays); Hyrje në trajtimin e përjashtimeve (Exception handling)	54
8.1 Hyrje	54
8.2 Vargjet	54
8.3 Deklarimi dhe krijimi i vargjeve	55
8.4 Shembuj që përdorin vargje	56
8.4.4 Mbledhja si shumë e elementeve të një vargu	59
8.4.5 Iterimi nëpër vargje me <i>foreach</i>	59
8.4.6 Përdorimi i bar charts (grafikëve me shirita) për t'i shfaqur grafikisht shënimet e vargut; type inference (nxjerrja e tipit) me <i>var</i>	60
8.4.7 Përdorimi i elementeve të një vargu si numërues	62
8.5 Përdorimi i vargjeve për t'i analizuar rezultatet e sondazhit; Hyrje në trajtimin e përjashtimeve (exception handling)	63
8.5.2 Trajtimi i përjashtimeve: Procesimi i përgjigjes së parregullt	64
8.5.3 Urdhri <i>try</i>	64
8.5.3 Ekzekutimi i bllokut <i>catch</i>	64
8.5.5 Properti <i>Message</i> e parametrat të përjashtimit	65
8.6 Studim i rastit: Simulimi i përzierjes dhe (shpër)ndarjes së letrave të lojës (kartave)	65
8.6.1 Klasa Leter dhe Properti-t Getter-Only (Vetëm Marrëse) të implementuara automatikisht (auto-implemented)	65

8.7 Pasimi (dërgimi, bartja) e vargjeve dhe elementeve të vargjeve te metodat	68
8.8 Studim i rastit: ditari i notave që e përdor një varg për t'i ruajtur notat.....	70
8.9 Vargjet shumë-dimensionale	74
8.10 Studim i rastit: DitarINotave duke përdorur varg drejtkëndësh.....	77
8.11 Listat e argumenteve me gjatësi të ndryshueshme (variable, variable-length)	83
8.12 Përdorimi i argumenteve command-line (në dritaren komanduese).....	84
8.13 (Opsionale) Pasimi i vargjeve me vlerë dhe me referencë	86
9. Hyrje në LINQ dhe në koleksionin List	89
9.1 Hyrje	89
9.2 Bërja e query-ve në një varg të vlerave int duke përdorur LINQ.....	89
9.2.1 Klauzola from	91
9.2.2 Klauzola where.....	91
9.2.3 Klauzola select.....	92
9.2.4 Iterimi nëpër rezultatet e query-t LINQ	92
9.2.5 Klauzola orderby	92
9.2.6 Interfejsi (ndërfaqja) IEnumerable<T>.....	92
9.3 Bërja e query-ve në një varg të objekteve Punonjës duke përdorur LINQ.....	93
9.3.2 Sortimi i rezultateve të një query LINQ sipas shumë properti-ve.....	96
9.3.3 Metodot e zgjerimit Any, First dhe Count.....	96
9.3.4 Zgjedhja e një properti-e të një objekti	96
9.3.5 Krijimi i tipeve të reja në klauzolën select të një query LINQ	96
9.4 Hyrje në koleksione.....	97
9.4.1 Koleksioni List<T>.....	97
9.4.2 Ndryshimi dinamik i madhësisë së koleksionit List<T>	99
9.5 Bërja e query-ve në koleksionin gjenerik List duke përdorur LINQ	101
9.5.1 Klauzola let	102
9.5.2 Ekzekutimi i shtyrë për më vonë.....	102
9.5.3 Metodot e zgjerimit ToArray dhe ToList	103
9.5.4 Inicializuesit e koleksioneve.....	103
10. Klasat dhe objektet: vështrim më i thellë	104
10.2 Studim i rastit i klasës <i>Time</i> ; hedhja e përjashtimeve.....	104
10.2.1 Deklarimi i klasës <i>Time1</i>	104
10.2.2 Përdorimi i klasës <i>Time1</i>	105
10.3 Kontrollimi i qasjes tek anëtarët.....	106

10.4 Referimi tek anëtarët e objektit aktual me referencën <i>this</i>	108
10.5 Konstruktoret e mbingarkuar	110
10.6 Konstruktoret default (të paracaktuar) dhe pa parametra	114
10.7 Kompozimi (Composition)	115
10.8 Garbage collection (mbledhja e mbeturinave) dhe destruktoret (shkatërruesit)	117
10.9 Anëtarët <i>static</i> -ë të klasës	118
10.10 Variablat e instancës që janë <i>readonly</i> (vetëm për lexim)	120
10.11 Class View (Pamja e klasave) dhe Object Browser (Shfletuesi i Objekteve)	120
10.12 Inicializuesit e objekteve	122
10.13 Mbingarkimi i operatorëve; Prezentimi i <i>struct</i>	122
10.13.1 Krijimi i tipeve vlerë me <i>struct</i>	122
10.13.2 Tipi vlerë <i>ComplexNumber</i> (NumërKompleks)	123
10.14 Studimi i rastit Klasa Kohë: metodat zgjeruese (Extension Methods)	125
11. Programimi i Orientuar në Objekte: Trashëgimia	127
11.1 Hyrje	127
11.2 Klasat bazë dhe klasat e derivuara (të prejardhura, të nxjerra)	127
11.3 Anëtarët e mbrojtur (<i>protected</i>)	128
11.4. Relacioni ndërmjet klasave bazë dhe klasave të derivuara	129
11.4.1 Krijimi dhe përdorimi i një klase	129
11.4.2 Krijimi i një klase pa e përdorur trashëgiminë	133
11.4.3 Krijimi i një hierarkie të trashëgimisë	136
11.4.4 Hierarkia e trashëgimisë duke përdorur variabla të instancës që janë <i>protected</i>	140
11.4.5 Hierarkia e trashëgimisë duke përdorur variabla të instancës që janë <i>private</i>	145
11.5 Konstruktoret në klasat e derivuara	149
11.6 Inxhinierimi i softuerit me trashëgimi	149
11.7 Klasa <i>object</i>	149
12. Programimi i Orientuar në Objekte: Polimorfizmi dhe Interfejsat (Shumëformësia dhe Ndërfaqet)	
.....	151
12.1 Hyrje	151
12.2 Shembuj të polimorfizmit	151
12.3 Demonstrimi i sjelljes polimorfike	151
12.4 Klasat dhe metodat abstrakte	156
12.5 Studim i rastit: Sistemi i pagave duke përdorur polimorfizëm	157
12.6 Metodot dhe klasat sealed (të vulosura)	168

12.7 Studim i rastit: Krijimi dhe përdorimi i interfejsave (ndërfaqeve).....	169
Bibliografia	177
Online	177
Licensimi.....	177

4. Hyrje në klasa, objekte, metoda dhe stringje

4.1 Hyrje

- Çdo klasë që e krijoni bëhet një tip (lloj) i ri që mund ta përdorni për të krijuar objekte, prandaj C# është gjuhë e zgjerueshme e programimit.

4.2 Testimi i një klase *Llogari (Account)*

```
// Fig. 4.1: TestiLlogarise.cs
// Krijimi dhe manipulimi i një objekti LlogariEBankes.
using System;

class TestiLlogarise
{
    static void Main()
    {
        // krijoje një objekt LlogariEBankes dhe caktoja atë variablës llogariaIme
        LlogariEBankes llogariaIme = new LlogariEBankes();

        // shfaqje emrin fillestar të ndryshores llogariaIme
        // (fillimisht nuk ka emër)
        Console.WriteLine($"Emri fillestar është: {llogariaIme.MerreEmrin()}");

        // kërkoje nga përdoruesi dhe lexoje emrin, pastaj vendose emrin në objekt
        Console.Write("Shkruaje emrin: "); // kërkesa apo inkurajimi
        string emri = Console.ReadLine(); // lexoje emrin
        llogariaIme.CaktojeEmrin(emri); // vendose emrin në objektin llogariaIme

        // shfaqje emrin e vendosur në objektin llogariaIme
        Console.WriteLine(
            $"emri i objektit llogariaIme është: {llogariaIme.MerreEmrin()}");

        Console.ReadLine();
    }
}
```

```
Emri fillestar është:
Shkruaje emrin: Filan Fisteku
emri i objektit llogariaIme është: Filan Fisteku
_
```

```
// Fig. 4.2: LlogariEBankes.cs
// Klasë e thjeshtë LlogariEBankes që e përmban
// një variabël (ndryshore) të instancës 'emri'
// dhe metodat publike për ta Caktuar (Set) dhe Marrë (Get)
// vlerën e emrit

class LlogariEBankes
{
    private string emri; // variabël e instancës

    // metodë që e cakton emrin e llogarisë në objekt
    public void CaktojeEmrin(string emriLlogarise)
    {
        emri = emriLlogarise; // ruaje emrin e llogarisë
    }

    // metodë që e merr emrin e llogarisë nga objekti
    public string MerreEmrin()
    {
        return emri; // ia kthen vlerën e emrit - thirrësit të kësaj metode
    }
}
```

- Klasat nuk mund të ekzekutohen vetë.
- Një klasë që e përmban një metodë që i teston aftësitë e një klase tjetër quhet klasë **driver** (drejtuese/vojitëse).

4.2.1 Instancimi i një Objekti – Fjala kyçe *new* dhe konstruktorët

- Zakonisht, nuk mund ta thirrni një metodë të një klase deri kur ta krijoni një objekt të asaj klase.
- Një shprehje e krijimit të objektit e përdor operatorin **new** për ta krijuar një objekt.

4.2.2 Thirrja e metodës *GetName (MerreEmrin)* të klasës *Account (LlogariEBankes)*

- Për ta thirrur një metodë për një objekt specifik, e specifikoni emrin e objektit, të pasuar nga operatori i qasjes së anëtarit (.), pastaj emrin e metodës dhe një çift (set, grup) të kllapave. Kllapat e zbrazëta (bosh) tregojnë se metoda nuk kërkon ndonjë informacion shtesë për ta kryer detyrën e saj.
- Kur thirret një metodë, aplikacioni e bart (transferon) ekzekutimin nga thirrja e metodës te deklarimi i metodës, metoda e kryen detyrën e saj, pastaj kontrolli kthehet te pika e thirrjes së metodës.

4.2.3 Futja e një emri nga përdoruesi

- Metoda **ReadLine** e **Console**-s lexon karaktere deri kur ta hasë rreshtin e ri (newline), pastaj e kthen një **string** që i përmban karakteret deri te, por pa e përfshirë, rreshtin e ri, i cili shpërfillet.

4.2.4 Thirrja e metodës *Set* të klasës

- Një thirrje e metodës mund të ofrojë argumente që i ndihmojnë metodës ta kryejë detyrën e saj. Argumentet i vendosni në kllapat e thirrjes së metodës.

4.3 Klasa me ndryshore të instancës dhe metoda *Set* dhe *Get*

- Fakti që mund ta krijoni dhe ta manipuloni një objekt të një klase pa i ditur detalet e implementimit të klasës quhet abstraksion.

4.3.1 Deklarimi i klasës

- Ndryshoret e instancës të një klase i ruajnë shënimet për secilin objekt të klasës.

4.3.2 Fjala kyçe *class* dhe trupi i klasës

- Çdo deklarim i klasës e përmban fjalën kyçe **class** që pasohet menjëherë nga emri i klasës.
- Çdo deklarim i klasës zakonisht ruhet në një fajll që e ka emrin e njëjtë si të klasës dhe që mbaron me shtesën (ekstensionin) e fajllit **.cs**
- Trupi i klasës është i mbyllur në një çift të klasave gjarpërore { }
- Emrat e klasës, të properti-t (vetisë), të metodës dhe të ndryshoreve janë të gjithë identifikues dhe sipas marrëveshjes (konventës) të gjithë e përdorin skemën e emërimit **camel-case**. Emrat e klasave, të property-ve dhe të metodave nisin me shkronjën e parë të madhe dhe emrat e ndryshoreve nisin me shkronjën e parë të vogël.

4.3.3 Variabla (ndryshorja) e instancës e tipit string

- Çdo objekt e ka kopjen e vet të variablave të instancës së klasës.
- Variablat e instancës ekzistojnë para se një aplikacion t'i thërrasë metodat apo t'i qaset properti-ve (vetive) të një objekti, gjersa metodat dhe properti-t janë duke u ekzekutuar, dhe pasi ta kryejnë ekzekutimin metodat dhe properti-t.
- Variablat e instancës deklarohen brenda një deklarimi të klasës por jashtë trupave të metodave dhe properti-ve të klasës.
- Klientët e një klase – domethënë çdo kod tjetër që i thërret metodat e klasës (apo i qaset properti-ve të saj) – nuk mund t'u qasen variablave **private** të instancës. Mirëpo, klientët mund t'u qasen metodave (dhe propertive) **publike** të një klase.
- Çdo variabël (ndryshore) e instancës e ka një vlerë fillestare të paracaktuar (default) – një vlerë të ofruar nga C# kur nuk e specifikoni vlerën fillestare të variablës së instancës.
- Variablat e instancës nuk kërkohet të inicializohen në mënyrë eksplicite para se të përdoren në program – pos nëse duhet të inicializohen me vlera të ndryshme nga vlerat e tyre të paracaktuara (default).
- Vlera e paracaktuar për një variabël të instancës të tipit **string** është **null**.
- Kur e shfaqni vlerën e një variable string që e përmban vlerën null, në ekran nuk shfaqet kurrfarë teksti.

4.3.4 Metoda SetName

- Tipi i kthimit **void** tregon se një metodë nuk i kthen asnjë informacion thirrësit të vet.
- Një metodë mund të kërkojë një apo më shumë parametra që i paraqesin shënimet që i duhen për ta kryer detyrën e saj.
- Parametrat deklarohen në një listë të parametrave të vendosur në kllapat e kërkuara pas emrit të metodës.
- Çdo parametër *duhet* ta specifikojë një tip të pasuar nga një emër i parametrit. Kur ka shumë parametra, secili ndahet nga vijuesi me një presje.

- Çdo vlerë e argumentit në kllapat e thirrjes së metodës kopjohet në parametrin përkatës të deklarimit të metodës.
- Numri dhe radha e argumenteve në një thirrje të metodës duhet të përputhet me numrin dhe radhën e parametrave në listën e parametrave të deklarimit të metodës.
- Çdo trup i metodës kufizohet nga një kllapë gjarpërore e hapur dhe një kllapë gjarpërore e mbyllur. Brenda kllapave janë një apo më shumë urdhra që kryejnë detyrat e metodës.
- Variablat e deklaruar në trupin e një metode të caktuar janë variabla lokale që mund të përdoren vetëm në atë metodë. Parametrat e një metode janë po ashtu variabla lokale të asaj metode.

4.3.5 Metoda GetName

- Kur një metodë me tip të kthimit të ndryshëm nga **void** thirret dhe e kryen detyrën e vet, ajo duhet t'ia kthejë një rezultat thirrësit të vet përmes një urdhri **return**.

4.3.6 Modifikuesit (ndryshuesit) e qasjes *private* dhe *public*

- Fjala kyçe **private** është modifikues i aksesit (ndryshues i qasjes).
- Një variabël e instancës që është **private** është e qasshme vetëm për metodat (dhe anëtarët e tjerë, si properti-t) e klasës së variablës së instancës. Kjo njihet si fshehje e informatës.
- Shumica e variablave të instancës deklarohen **private**.
- Metodat (dhe anëtarët e tjerë të klasës) që janë të deklaruar **public** (publike) mund të përdoren nga metodat (dhe anëtarët e tjerë) të klasës në të cilën janë deklaruar, dhe nga klientët e klasës.
- Si rregull e paracaktuar (default), anëtarët e klasës janë **private** (privatë), pos nëse i specifikoni ndryshe përmes modifikuesve të qasjes.

4.3.6 Diagrami i klasës në UML (Unified Modeling Language – Gjuha e unifikuar e modelimit)

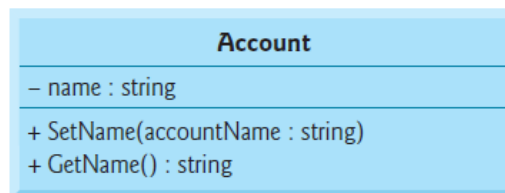


Fig. 4.3 | UML class diagram for class Account of Fig. 4.2.

- Diagramet e klasave në UML i përmbledhin atributet dhe operacionet e klasës.
- Diagramet UML u ndihmojnë dizajguesve (projektuesve) të sistemeve t'i specifikojnë sistemet në mënyrë koncize, grafike, të pavarur nga gjuha e programimit, para se programuesit t'i implementojnë sistemet në gjuhë specifike të programimit.
- Në UML, çdo klasë modelohet në një diagram të klasës si drejtkëndësh me tri ndarje.
- Ndarja e sipërme e përmban emrin e klasës të qendëruar horizontalisht me shkronja bold (të trasha).
- Ndarja e mesme i përmban atributet e klasës.
- Një shenjë e minusit (–) si modifikues i qasjes dhe emri i atributit tregon që atributi është **private** (privat).
- Pas emrit të atributit janë dy pika dhe tipi i atributit.
- Ndarja e poshtme i përmban operacionet (veprimet) e klasës.
- UML i modelon operacionet duke e listuar emrin e operacionit të paraprirë nga një modifikues i qasjes.
- Një shenjë e plusit (+) e tregon një operacion publik.
- UML e tregon tipin e kthimit të një operacioni duke e vendosur një dypikësh dhe tipin e kthimit pas kllapave që e pasojnë emrin e operacionit.
- UML e modelon një parametër duke e listuar emrin e parametrit, të pasuar nga një dypikësh dhe tipin e parametrit në kllapa pas emrit të operacionit.

4.4 Krijimi, kompajlimi dhe ekzekutimi i një projekti me dy klasa

- Mund ta hapni çdo klasë në editorin e Visual Studios duke klikuar dy herë në emrin e fajllit në dritaren **Solution Explorer**.
- Kur zgjedhni **Build > Build Solution** në Visual Studio, IDE i kompajlon të gjithë fajllat në projekt për ta krijuar aplikacionin e ekzekutueshëm.

4.5-6 Inxhinierimi i softuerit me metodat Set dhe Get, Property

```
// Fig. 4.5: TestILlogariseSeBankes.cs
// Krijimi dhe manipulimi i një objekti LlogariEBankes me properties (veti)
using System;

class TestILlogariseSeBankes
{
    static void Main()
    {
        // krijoje një objekt LlogariEBankes dhe caktoja ndryshores llogariaIme
        LlogariEBankes llogariaIme = new LlogariEBankes();

        // shfaqe emrin fillestar të objektit llogariaIme
        Console.WriteLine($"Emri fillestar është: {llogariaIme.Emri}");

        // kërkoje dhe lexoje emrin, pastaj vendose emrin në objekt
        Console.Write("Të lutem shkruaje emrin: "); // kërkoje (inkurajoje)
        string emri = Console.ReadLine(); // lexoje një rresht të tekstit
        llogariaIme.Emri = emri; // vendose emri-n në Emri-n e objektit llogariaIme

        // shfaqe emrin e ruajtur në objektin llogariaIme
        Console.WriteLine($"emri i objektit llogariaIme është: {llogariaIme.Emri}");
        Console.ReadLine();
    }
}
```

```
Emri fillestar është:
Të lutem shkruaje emrin: Filan Fisteku
emri i objektit llogariaIme është: Filan Fisteku
```

```
// Fig. 4.6: LlogariEBankes.cs
// Klasa LlogariEBankes që i zëvendëson
// metodat publike CaktojeEmrin dhe MerreEmrin
// me një property (veti) publike Emri

class LlogariEBankes
{
    private string emri; // variabël e instancës

    // property (veti) për ta caktuar dhe marrë variablën e instancës 'emri'
    public string Emri
    {
        get // e kthen vlerën e variablës përkatëse (korresponduese) të instancës
        {
            return emri; // ia kthen vlerën e emrit – kodit klient
        }
        set // ia cakton një vlerë të re variablës përkatëse (korresponduese) të instancës
        {
            emri = value; // value (vlera) është e deklaruar dhe inicializuar në mënyrë implicite
        }
    }
}
```

- Metodat Set dhe Get mund t'i validojnë tentimet për t'i ndryshuar shënimet **private** dhe ta kontrollojnë se si i paraqiten ato shënime thirrësit, përkatësisht.
- Një **property** (veti) e përmban një aksesor (qasës) **set** për ta ruajtur një vlerë në një variabël dhe një aksesor **get** për ta marrë vlerën e variablës.
- Për t'iu qasur një properti-e (vetie) për t'ia marrë vlerën, specifikoje emrin e objektit, pasuar nga operatori i qasjes së anëtarit (.) dhe emri i properti-t – kjo e ekzekuton në mënyrë implicite aksesoren get të properti-t, pos nëse shprehja është në anën e majtë të caktimit të vlerës (assignment).
- Qasja e një properti-t në të majtë të caktimit të vlerës e ekzekuton në mënyrë implicite aksesoren (qasësin) **set** të properti-t.

4.6.2 Klasa me variabël të instancës dhe një properti

- Deklarimi i properti-t e specifikon specifikuesin e qasjes, tipin, emrin dhe trupin e properti-t (vetisë)
- Me konventë (sipas marrëveshjes), identifikuesi i properti-t është identifikuesi që nis me shkronjë të madhe i variablës së instancës të cilën e manipulon.
- Një **get** accessor (qasës për marrje) nis me fjalën kyçe kontekstuale **get** pasuar nga trupi, që e përmban një urdhër **return** që e kthen vlerën e një variable përkatëse të instancës.
- Notacioni i properti-t ia lejon klientit të mendojë për properti-n si shënim themelor.
- Një **set** accessor (qasës për caktim të vlerës) nis me fjalën kyçe kontekstuale **set** pasuar nga trupi i vet.
- Fjala kyçe kontekstuale **value** e qasësit **set** deklarohet dhe inicializohet në mënyrë implicite për ju me vlerën të cilën kodi klient ia cakton properti-t.

4.6.3 Diagrami UML i klasës me një properti

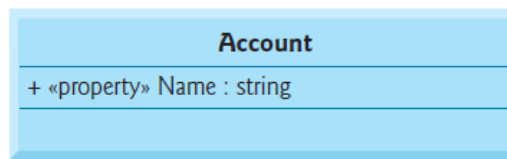


Fig. 4.7 | UML class diagram for class Account of Fig. 4.6.

- Properti-t e C# modelohen në UML si atributet. Një properti publike tregohet me simbolin plus (+) të pasuar nga fjala “property” në thonjëza guillemets (« dhe »), pastaj emri i properti-t, një dypikësh dhe tipi i properti-t.
- Përdorja e fjalëve përshkruese në thonjëzat guillemets (të quajtura stereotype në UML) ndihmon të dallohen properti-t nga atributet dhe operacionet e tjera.
- Një diagram i klasës ju ndihmon ta dizajnoni një klasë, kështu që nuk kërkohet ta shfaqë çdo detal të implementimit.
- Meqë një variabël e instancës që manipulohet nga një properti është realisht detal i implementimit i asaj properti-e, diagrami i klasës nuk e shfaq variablën përkatëse të instancës për properti-n.
- Ngjashëm, aksesoret (qasësit, accessors) **get** dhe **set** të një properti-e janë detale të implementimit, prandaj nuk listohen në diagram UML.

4.7 Properti-t e implementuara automatikisht (auto-implemented)

- Për properti-t në të cilat aksessori (qasësi) **get** thjesht e kthen vlerën e një variable të instancës dhe një aksesor **set** thjesht ia cakton një vlerë variablës së instancës, C# i ofron properti-t e implementuara automatikisht.
- Me properti-n e implementuar automatikisht, kompajleri i C# e krijon një variabël të fshehur të instancës, dhe aksesoret **get** dhe **set** për ta marrë dhe për ta caktuar atë variabël të instancës.

4.8 Inicializimi i objekteve me konstruktorë

```
// Fig. 4.8: LlogariEBankes.cs
// Klasa LlogariEBankes me një konstruktor që e inicializon
// emrin e një objekti LlogariEBankes

class LlogariEBankes
{
    public string Emri { get; set; } // veti e implementuar automatikisht

    // konstruktori (ndërtuesi) e cakton properti-n (vetinë) Emri
    // në vlerën e parametrin emriILlogarise
    public LlogariEBankes(string emriILlogarise) // emri i konstruktorit është emri i klasës
    {
        Emri = emriILlogarise;
    }
}
```

- Çdo klasë që e deklaroni, opsionalisht mund ta ofrojë një konstruktor me parametra që mund të përdoret për ta inicializuar një objekt kur të krijohet.
- C# e kërkon një thirrje të konstruktorit për çdo objekt që krijohet, kështu që kjo është pika ideale për t'i inicializuar variablat e instancës së objektit.

4.8.1 Deklarimi i konstruktorit për inicializim specifik të objektit

- Konstruktori (ndërtuesi) duhet ta ketë emrin e njëjtë si klasa e tij.
- Një dallim i rëndësishëm ndërmjet konstruktorëve dhe metodave është se konstruktorët nuk mund ta specifikojnë një tip të kthimit (madje as **void**).
- Normalisht, konstruktorët deklarohen **public** që të mund të përdoren nga kodi klient i klasës për t'i inicializuar objektet e klasës.

4.8.2 Inicializimi i objekteve kur krijohen

```
// Fig. 4.9: TestILogariseSeBankes.cs
// Përdorja e konstruktorit LlogariEBankes
// për ta caktuar emrin e objektit LlogariEBankes
// kur krijohet objekti LlogariEBankes
using System;

class TestILogariseSeBankes
{
    static void Main()
    {
        // krijoji dy objekte LlogariEBankes
        LlogariEBankes llogaria1 = new LlogariEBankes("Jane Green");
        LlogariEBankes llogaria2 = new LlogariEBankes("John Blue");

        // shfaq vlerën fillestare të emrit për secilën LlogariEBankes
        Console.WriteLine($"emri i objektit llogaria1 është: {llogaria1.Emri}");
        Console.WriteLine($"emri i objektit llogaria2 është: {llogaria2.Emri}");
        Console.ReadLine();
    }
}
```

```
emri i objektit llogaria1 është: Jane Green
emri i objektit llogaria2 është: John Blue
```

- Krijimi i një objekti me **new** e thërret në mënyrë implicite konstruktorin e klasës për ta inicializuar objektin.
- Në çdo klasë që nuk e deklaron në mënyrë eksplicite një konstruktor, kompajleri e ofron një konstruktor të paracaktuar **default** pa parametra.
- Kur një klasë e ka vetëm konstruktorin default, variablat e instancës së klasës inicializohen me vlerat e tyre të paracaktuara – 0 për tipet e thjeshta numerike, **false** për tipin e thjeshtë **bool** dhe **null** për të gjitha tipet e tjera.
- Nëse e deklaroni një apo më shumë konstruktorë për një klasë, kompajleri nuk do të krijojë konstruktor **default**.

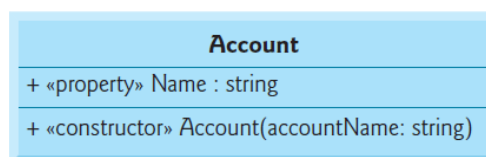


Fig. 4.10 | UML class diagram for Account class of Fig. 4.8.

- UML i modelon konstruktorët si operacione në ndarjen e tretë të diagramit të klasës. Për ta dalluar një konstruktor nga operacionet e tjera të klasës, UML kërkon që fjala “constructor” të futet në thonjëza guillemets (« dhe ») dhe të vendoset para emrit të konstruktorit.
- Është e zakonshme të listohen konstruktorët para operacioneve të tjera në ndarjen e tretë.

4.9 Klasa me variabël *decimal* të instancës

```
// Fig. 4.11: LlogariEBankes.cs
// Klasa LlogariEBankes me një balans dhe me një metodë Depozito

class LlogariEBankes
{
    public string Emri { get; set; } // properti (veti) e implementuar
    automatikisht
    private decimal balansi; // variabël e instancës

    // Konstruktori LlogariEBankes që i pranon dy parametra
    public LlogariEBankes(string emriILlogarisë, decimal balansiFillestar)
    {
        Emri = emriILlogarisë;
        Balansi = balansiFillestar; // aksesori (qasësi) set i Balansit e validon
    }

    // Properti (vetia) Balansi me validim
    public decimal Balansi
    {
        get
        {
            return balansi;
        }
        private set // mund të përdoret vetëm brenda klasës
        {
            // valido se balansi është më i madh se 0.0;
            // nëse nuk është, variabla e instancës balansi
            // e mban vlerën e vet paraprake
            if (value > 0.0m) // m tregon se 0.0 është literal decimal
            {
                balansi = value;
            }
        }
    }

    // metodë që i depoziton (shton) balansit vetëm shumë valide
    public void Depozito(decimal sasiaEDepozitimit)
    {
        if (sasiaEDepozitimit > 0.0m) // nëse sasiaEDepozitimit është valide
        {
            Balansi = Balansi + sasiaEDepozitimit; // shtoja balansit
        }
    }
}
```

- Tipi **decimal** është dizajnuar t'i paraqesë në mënyrë precize numrat me pikë decimale, siç janë vlerat monetare.
- Një variabël decimale e instancës inicializohet në zero by default (në mënyrë të paracaktuar).
- Një aksesori **set** i një properti-e mund të bëjë validim (i njohur edhe si validity checking, kontrollimi i vlefshmërisë).
- Shkronja **m** e shtuar pas një literalit numerik tregon se numri është literal **decimal**.
- By default (në mënyrë të paracaktuar), aksesoret **get** dhe **set** të një properti-e e kanë qasjen e njëjtë si properti.
- Aksesoret **get** dhe **set** mund të kenë modifikues të ndryshëm të qasjes. Një nga aksesoret duhet të ketë në mënyrë implicite qasje të njëjtë si properti dhe tjetri duhet të deklarohet me qasje më restriktive (më të kufizuar).
- Aksesori **private set** i një properti-e mund të përdoret vetëm nga klasa e properti-t, e jo nga klientët e klasës.

4.9.2 Klasa që krijon dhe përdor objekte

```
// Fig. 4.12: TestILlogariseSeBankes.cs
// Leximi dhe shkrimi i vlerave monetare
// me objekte LlogariEBankes
using System;

class TestILlogariseSeBankes
{
    static void Main()
    {
        LlogariEBankes llogaria1 = new LlogariEBankes("Jane Green", 50.00m);
        LlogariEBankes llogaria2 = new LlogariEBankes("John Blue", -7.53m);

        // shfaq balansin fillestar të secilit objekt
        Console.WriteLine(
            $"balansi për {llogaria1.Emri}: {llogaria1.Balansi:C}");
        Console.WriteLine(
            $"balansi për {llogaria2.Emri}: {llogaria2.Balansi:C}");

        // kërkoje pastaj lexoje hyrjen
        Console.Write("\nShkruaje shumën e depozitës për llogarinë1: ");
        decimal shumaEDepozitës = decimal.Parse(Console.ReadLine());
        Console.WriteLine(
            $"po i shtoj {shumaEDepozitës:C} në balansin e llogarisë1\n");
        llogaria1.Depozito(shumaEDepozitës); // shtoja balansit të llogarisë1

        // shfaq balanset
        Console.WriteLine(
            $"balansi për {llogaria1.Emri}: {llogaria1.Balansi:C}");
        Console.WriteLine(
            $"balansi për {llogaria2.Emri}: {llogaria2.Balansi:C}");

        // kërkoje pastaj lexoje hyrjen
        Console.Write("\nShkruaje shumën e depozitës për llogarinë2: ");
        shumaEDepozitës = decimal.Parse(Console.ReadLine());
        Console.WriteLine(
            $"po i shtoj {shumaEDepozitës:C} në balansin e llogarisë2\n");
        llogaria2.Depozito(shumaEDepozitës); // shtoja balansit të llogarisë2

        // shfaq balanset
        Console.WriteLine(
            $"balansi për {llogaria1.Emri}: {llogaria1.Balansi:C}");
        Console.WriteLine(
            $"balansi për {llogaria2.Emri}: {llogaria2.Balansi:C}");
        Console.ReadLine();
    }
}
```

```
balansi për Jane Green: £50.00
balansi për John Blue: £0.00

Shkruaje shumën e depozitës për llogarinë1: 35
po i shtoj £35.00 në balansin e llogarisë1

balansi për Jane Green: £85.00
balansi për John Blue: £0.00

Shkruaje shumën e depozitës për llogarinë2: 75
po i shtoj £75.00 në balansin e llogarisë2

balansi për Jane Green: £85.00
balansi për John Blue: £75.00
```

Format specifier	Description
C or c	Formats the <code>string</code> as currency. Includes an appropriate currency symbol (\$) in the U.S.) next to the number. Separates digits with an appropriate <i>separator character</i> (in the U.S. its a comma between every three digits for thousands, millions, etc.) and sets the number of decimal places to two by default.
D or d	Formats the <code>string</code> as a whole number (integer types only).
N or n	Formats the <code>string</code> with a thousands separator and a default of two decimal places.
E or e	Formats the number using scientific notation with a default of six decimal places.
F or f	Formats the <code>string</code> with a fixed number of decimal places (two by default).
G or g	Formats the number normally with decimal places or using scientific notation, depending on context. If a format item does not contain a format specifier, format G is assumed implicitly.
X or x	Formats the <code>string</code> as hexadecimal (base 16 numbers; we discuss these in the online Number Systems appendix).

Fig. 4.13 | `string` format specifiers.

- Formatimin në një shprehje të interpolimit të stringut në C# 6 mund ta specifikoni duke e pasuar vlerën në kllapa gjarpërore { } me një dypikësh dhe një specifikues të formatit (format specifier).
- Specifikuesi i formatit **C** e formaton një vlerë numerike si valutë/monedhë (**C** është shkurtesë për currency = valutë) – zakonisht me dy shifra pas pikës decimale.
- Settings-at (Cilësimet) e Windows culture (në sistemin operativ) në makinën (kompjuterin) e përdoruesit e përcaktojnë formatin për shfaqjen e vlerave monetare, siç janë presjet apo pikat për ndarjen e mijësheve, milionësheve etj.
- Për dallim nga variablat e instancës, variablat lokale nuk inicializohen by default (në mënyrë të paracaktuar).
- Metoda **Parse** e tipit **decimal** e konverton (shndërron) një **string** në vlerë **decimal**-e.

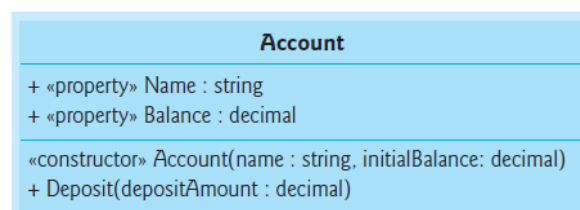


Fig. 4.14 | UML class diagram for `Account` class of Fig. 4.11.

7. Metodat: vështrim më i thellë

7.1 Hyrje

- Përvoja ka treguar se mënyra më e mirë për ta zhvilluar dhe mirëmbajtur një aplikacion të madh është ta ndërtojmë me pjesë të vogla e të thjeshta. Kjo teknikë quhet ndaj e mposht (përçaj e sundo, divide and conquer, divide et impera).

7.2 Paketimi i kodit në C#

- Tri mënyra të zakonshme të paketimit të kodit janë metodat, klasat dhe namespace-at (hapësirat e emrave).

7.2.1 Modularizimi i programeve

- Metodat ju lejojnë ta modularizoni një aplikacion duke i ndarë detyrat e tij në njësi të pavarura (self-contained, të plota dhe të afta të punojnë pa ndihmën e asgjëje tjetër).
- Ndarja e një aplikacioni në metoda kuptimplota e bën më të lehtë debug-imin (korrigjimin, heqjen e defekteve) dhe mirëmbajtjen.

7.3 Metodatat *static*-e, variablat *static*-e dhe klasa *Math*

Method	Description	Example
<code>Abs(x)</code>	absolute value of x	<code>Abs(23.7)</code> is 23.7 <code>Abs(0.0)</code> is 0.0 <code>Abs(-23.7)</code> is 23.7
<code>Ceiling(x)</code>	rounds x to the smallest integer not less than x	<code>Ceiling(9.2)</code> is 10.0 <code>Ceiling(-9.8)</code> is -9.0
<code>Floor(x)</code>	rounds x to the largest integer not greater than x	<code>Floor(9.2)</code> is 9.0 <code>Floor(-9.8)</code> is -10.0
<code>Cos(x)</code>	trigonometric cosine of x (x in radians)	<code>Cos(0.0)</code> is 1.0
<code>Sin(x)</code>	trigonometric sine of x (x in radians)	<code>Sin(0.0)</code> is 0.0
<code>Tan(x)</code>	trigonometric tangent of x (x in radians)	<code>Tan(0.0)</code> is 0.0

Fig. 7.2 | *Math* class methods. (Part 1 of 2.)

Method	Description	Example
<code>Exp(x)</code>	exponential method e^x	<code>Exp(1.0)</code> is 2.71828 <code>Exp(2.0)</code> is 7.38906
<code>Log(x)</code>	natural logarithm of x (base e)	<code>Log(Math.E)</code> is 1.0 <code>Log(Math.E * Math.E)</code> is 2.0
<code>Max(x, y)</code>	larger value of x and y	<code>Max(2.3, 12.7)</code> is 12.7 <code>Max(-2.3, -12.7)</code> is -2.3
<code>Min(x, y)</code>	smaller value of x and y	<code>Min(2.3, 12.7)</code> is 2.3 <code>Min(-2.3, -12.7)</code> is -12.7
<code>Pow(x, y)</code>	x raised to the power y (i.e., x^y)	<code>Pow(2.0, 7.0)</code> is 128.0 <code>Pow(9.0, 0.5)</code> is 3.0
<code>Sqrt(x)</code>	square root of x	<code>Sqrt(900.0)</code> is 30.0

Fig. 7.2 | *Math* class methods. (Part 2 of 2.)

- Mund ta thirrni çfarëdo metode `static`-e duke e specifikuar emrin e klasës në të cilën është deklaruar ajo, të pasuar nga operatori i qasjes në anëtar (`.`) dhe emrin e metodës, si në vijim

EmriKlasës.EmriIMetodës(argumentet)

- Argumentet e metodës mund të jenë konstanta, variabla apo shprehje
- Konstanta deklarohet me fjalën kyçe **const** – vlera e saj nuk mund të ndryshohet pasi të jetë deklaruar konstanta.
- **Math.PI** (3.1415926535897931) është raporti i perimetrit të rrethit ndaj diametrit të tij.
- **Math.E** (2.7182818284590451) është vlera bazë për logaritmet natyrore.
- Kur çdo objekt e një klase e mirëmban kopjen e vet të një atributi, çdo objekt (instancë) e klasës ka instancë të veçantë të variablës. Kur krijohen objektet e një klase që përmban variabla **static**-e, të gjitha objektet e asaj klase e bashkë-ndajnë një kopje të variablave **static**-e të klasës.
- Bashkërisht variablat **static**-e dhe variablat e instancës i paraqesin fushat e një klase.
- Nëse e deklaroni më shumë se një metodë **Main** ndër të gjitha klasat e projektit tuaj, do t'ju duhet të tregoni se cila nga to dëshironi të jetë pika hyrëse (fillestare, nisëse) e aplikacionit. Këtë mund ta bëni duke klikuar në menynë **Project > [ProjectName] Properties...** dhe duke zgjedhur klasën që e përmban metodën **Main** që duhet të jetë pika hyrëse nga list-box-i **Startup object**.

7.4 Metodat me shumë parametra

```
// Fig. 7.3: GjetesiIMaksimumit.cs
// Metoda Maksimumi me tre parametra
using System;

class GjetesiIMaksimumit
{
    // merri tri vlera me pikë lëvizëse dhe përcaktoje vlerën maksimale
    static void Main()
    {
        // kërkoji dhe futi tri vlera me pikë lëvizëse
        Console.Write("Shkruaje vlerën e parë me pikë lëvizëse: ");
        double numri1 = double.Parse(Console.ReadLine());
        Console.Write("Shkruaje vlerën e dytë me pikë lëvizëse: ");
        double numri2 = double.Parse(Console.ReadLine());
        Console.Write("Shkruaje vlerën e tretë me pikë lëvizëse: ");
        double numri3 = double.Parse(Console.ReadLine());

        // përcaktoje maksimumin e tri vlerave
        double rezultati = Maksimumi(numri1, numri2, numri3);

        // shfaqe vlerën maksimale
        Console.WriteLine("Maksimumi është: " + rezultati);
        Console.ReadKey();
    }

    // e kthen maksimumin e tre parametrave të vet double
    static double Maksimumi(double x, double y, double z)
    {
        double vleraMaksimale = x; // supozo se x është më i madhi në fillim

        // përcaktoje se a është y më i madh se sa vleraMaksimale
        if (y > vleraMaksimale)
        {
            vleraMaksimale = y;
        }

        // përcaktoje se a është z më i madh se sa vleraMaksimale
        if (z > vleraMaksimale)
        {
            vleraMaksimale = z;
        }

        return vleraMaksimale;
    }
}
```

```
Shkruaje vlerën e parë me pikë lëvizëse: 5.6
Shkruaje vlerën e dytë me pikë lëvizëse: 4.3
Shkruaje vlerën e tretë me pikë lëvizëse: 8.7
Maksimumi është: 8.7
```

- Kur kemi shumë parametra, ata specifikohen si listë e ndarë me presje.
- Kur thirret një metodë, secili parametër inicializohet me vlerën e argumentit përkatës (korrespondues). Duhet të ketë nga një argument në thirrjen e metodës për secilin parametër të kërkuar në deklarin e metodës. Secili argument duhet të jetë në përputhje (konsistent) me tipin (llojin) e parametrin përkatës (korrespondues).
- Kur kontrolli i programit kthehet në pikën në aplikacion ku është thirrur një metodë, parametrat e metodës më nuk janë të qasshëm.
- Metodatat mund ta kthejnë të shumtën një vlerë; vlera e kthyer mund të jetë tip i vlerës që përmban shumë vlera (e implementuar si **struct**) apo referencë të një objekt që përmban shumë vlera.
- C# lejon që objektet **string** të krijohen duke bashkuar stringje më të vegjël në string-a më të mëdhenj duke e përdorur operatorin +. Kjo njihet si ngjitje e stringjeve (string concatenation).
- Çdo vlerë e një tipi të thjeshtë në C# e ka një paraqitje në formë të string-ut. Kur një operandët e operatorit + është string, tjetri shndërrohet (konvertohet) në mënyrë implicite në string, pastaj ata dy ngjiten.
- Të gjitha objektet e kanë një metodë **ToString** që e kthen një paraqitje string të objektit. Kur një objekt ngjitet me një string, metoda ToString e objektit thirret në mënyrë implicite për ta marrë paraqitjen string të objektit, pastaj dy stringjet ngjiten.

7.5 Informata detale për deklarimin dhe përdorimin e metodave

- I keni parë tri mënyra për ta thirrur një metodë:
 - duke e përdorur emrin e metodës të vetëm për ta thirrur një metodë tjetër të klasës së njëjtë;
 - duke e përdorur një variabël që e përmban një referencë të një objekt, të pasuar nga operatori i qasjes së anëtarit (.) dhe emrin e metodës për ta thirrur një metodë jo-statike të objektit të referencuar;
 - dhe duke e përdorur emrin e klasës dhe operatorin e qasjes së anëtarit (.) për ta thirrur një metodë statike të një klase.
- Një metodë statike mund ta thërrasë vetëm metoda të tjera statike të klasës së njëjtë dhe mund t'i manipulojë vetëm variablat statike në klasën e njëjtë.
- I keni parë tri mënyra për t'ia kthyer kontrollin urdhrit që e thirr metodën:
 - kur arrihet kllapa e djathtë gjarpërore } në një metodë me tip kthyes **void**,
 - kur ekzekutohet urdhri vijues në një metodë me tip kthyes **void**:

return;
 - dhe kur një metodë e kthen një rezultat me një urdhër të formës vijuese në të cilën *shprehja* vlerësohet (i llogaritet vlera) dhe rezultati i saj (dhe kontrolli) i kthehet thirrësit:

return shprehja;

7.6 Shndërrimi i tipeve të argumenteve

Type	Conversion types
bool	no possible implicit conversions to other simple types
byte	ushort, short, uint, int, ulong, long, decimal, float or double
char	ushort, int, uint, long, ulong, decimal, float or double
decimal	no possible implicit conversions to other simple types
double	no possible implicit conversions to other simple types
float	double
int	long, decimal, float or double
long	decimal, float or double
sbyte	short, int, long, decimal, float or double
short	int, long, decimal, float or double
uint	ulong, long, decimal, float or double
ulong	decimal, float or double
ushort	uint, int, ulong, long, decimal, float or double

Fig. 7.4 | Implicit conversions between simple types.

- Një veçori tjetër e rëndësishme e thirrjeve të metodave është shndërrimi i tipeve të argumenteve – konvertimi implicit i vlerës së një argumenti në tipin të cilin e pret metoda për ta pranuar në parametrin e saj përkatës.
- Rregullat e shndërrimit të tipit vlejnë për shprehjet që përmbajnë vlera të dy apo më tepër tipeve të thjeshta dhe për vlerat me tipe të thjeshta që u pasohen (dërgohen) metodave si argumente.
- Në rastet kur mund të humbet informacioni për shkak të shndërrimit (konvertimit) ndërmjet tipeve të thjeshta, kompajleri kërkon ta përdorni operatorin cast (të shndërrimit) për ta detyruar shndërrimin në mënyrë eksplicite.

7.7 .NET Framework Class Library (Libraria e klasave e .NET Framework)

- Shumë klasa të parafinuara janë të grupuara në kategori të klasave të ndërlidhura të quajtura namespace-a (hapësira të emrave). Bashkë këtyre namespace-ave u referohemi si .NET Framework Class Library (Libraria e klasave e .NET Framework).

Namespace	Description
System.Windows.Forms	Contains the classes required to create and manipulate GUIs. (Various classes in this namespace are discussed in Chapter 14, Graphical User Interfaces with Windows Forms: Part 1, and Chapter 15, Graphical User Interfaces with Windows Forms: Part 2.)
System.Windows.Controls System.Windows.Input System.Windows.Media System.Windows.Shapes	Contain the classes of the Windows Presentation Foundation for GUIs, 2-D and 3-D graphics, multimedia and animation. (See the online WPF chapters.)
System.Linq	Contains the classes that support Language Integrated Query (LINQ). (See Chapter 9, Introduction to LINQ and the List Collection, and several other chapters throughout the book.)
System.Data.Entity	Contains the classes for manipulating data in databases (i.e., organized collections of data), including support for LINQ to Entities. (See Chapter 22, Databases and LINQ.)
System.IO	Contains the classes that enable programs to input and output data. (See Chapter 17, Files and Streams.)
System.Web	Contains the classes used for creating and maintaining web apps, which are accessible over the Internet. (See the online ASP.NET Web App Development chapter.)
System.Xml	Contains the classes for creating and manipulating XML data. Data can be read from or written to XML files. (See the online XML and LINQ to XML chapter.)
System.Xml.Linq	Contains the classes that support Language Integrated Query (LINQ) for XML documents. (See the online XML and LINQ to XML chapter, and several other chapters throughout the book.)
System.Collections System.Collections.Generic	Contain the classes that define data structures for maintaining collections of data. (See Chapter 21, Generic Collections; Functional Programming with LINQ/PLINQ.)
System.Text	Contains classes that enable programs to manipulate characters and strings. (See Chapter 16, Strings and Characters: A Deeper Look.)

Fig. 7.5 | .NET Framework Class Library namespaces (a subset).

7.8 Studim i rastit: Gjenerimi (prodhimi) i numrave të rastësishëm

```
// Fig. 7.6: NumratEPloteTeRastesishem.cs
// Numrat e plotë të rastësishëm të ndryshëm dhe të shkallëzuar
using System;

class NumratEPloteTeRastesishem
{
    static void Main()
    {
        Random numratERastesishem = new Random(); // gjenerues i numrave të rastësishëm

        // sillu në cikël 20 herë
        for (int numeruesi = 1; numeruesi <= 20; ++numeruesi)
        {
            // zgjedhe një numër të plotë të rastësishëm nga 1 deri në 6
            int faqjaEKubit = numratERastesishem.Next(1, 7);
            Console.Write($"{faqjaEKubit} "); // shfaq vlerën e gjeneruar (prodhuar)
        }

        Console.WriteLine();
        Console.ReadLine();
    }
}
```



5 5 6 2 1 4 4 1 5 1 4 6 1 4 3 3 6 6 2 2

```
// Fig. 7.7: SilleZarin.cs
// Hidhe 60,000,000 herë një zar me gjashtë faqe.
using System;

class SilleZarin
{
    static void Main()
    {
        Random numratERastësishëm = new Random(); // gjenerator (prodhues) i numrave të rastësishëm

        int frekuenca1 = 0; // numri i 1shave të hedhur
        int frekuenca2 = 0; // numri i 2shave të hedhur
        int frekuenca3 = 0; // numri i 3shave të hedhur
        int frekuenca4 = 0; // numri i 4shave të hedhur
        int frekuenca5 = 0; // numri i 5shave të hedhur
        int frekuenca6 = 0; // numri i 6shave të hedhur

        // përmbledhi rezultatet e 60,000,000 hedhjeve të një zari
        for (int hedhja = 1; hedhja <= 60000000; ++hedhja)
        {
            int faqjaEZarit = numratERastësishëm.Next(1, 7); // numër nga 1 deri në 6

            // përcaktoje vlerën e hedhjes dhe rrite numëruesin e duhur
            switch (faqjaEZarit)
            {
                case 1:
                    ++frekuenca1; // rrite numëruesin e 1shave
                    break;
                case 2:
                    ++frekuenca2; // rrite numëruesin e 2shave
                    break;
                case 3:
                    ++frekuenca3; // rrite numëruesin e 3shave
                    break;
                case 4:
                    ++frekuenca4; // rrite numëruesin e 4shave
                    break;
                case 5:
                    ++frekuenca5; // rrite numëruesin e 5shave
                    break;
                case 6:
                    ++frekuenca6; // rrite numëruesin e 6shave
                    break;
            }
        }

        Console.WriteLine("Faqja\tFrekuenca"); // shfaq titujt
        Console.WriteLine($"1\t{frekuenca1}\n2\t{frekuenca2}");
        Console.WriteLine($"3\t{frekuenca3}\n4\t{frekuenca4}");
        Console.WriteLine($"5\t{frekuenca5}\n6\t{frekuenca6}");

        Console.ReadLine();
    }
}
```

Faqja	Frekuenca
1	9994093
2	9999810
3	9999092
4	10005682
5	10000394
6	10000929

- Metoda e rastësishme **Next()** e gjeneron një vlerë të rastësishme int në rangun 0 deri në +2,147,483,646, inkluzive (përfshirëse - duke i përfshirë vlerat kufitare).
- Metodat e klasës Random në fakt gjenerojnë numra pseudo-të-rastësishëm duke u bazuar në llogaritje komplekse matematike. Llogaritja që i prodhon numrat pseudo-të-rastësishëm e përdor kohën e ditës si vlerë të bërthamës (seed) për ta ndryshuar pikën fillestare të sekuencës.
- Klasa Random ofron versione të tjera të metodës Next. Njëri version e pranon një int dhe e kthen një vlerë nga 0 deri te, por pa e përfshirë, vlerën e argumentit. Versioni tjetër i pranon dy int-a dhe e kthen një vlerë nga vlera e argumentit të parë deri te, por pa e përfshirë, vlerën e argumentit të dytë.
- Nëse përdoret e njëjta vlerë bërthamë secilën herë, objekti Random e prodhon sekuencën e njëjtë të numrave të rastësishëm. Konstruktori i klasës Random mund ta pranojë vlerën e bërthamës si argument.

7.9 Studim i rastit: Një lojë e fatit; Hyrje në Enumeracione (Enumerations)

- Një enumeracion paraqitet me fjalën kyçe **enum** dhe një emër të tipit. Kllapat gjarpërore e kufizojnë trupin e një deklarimi enum. Brenda kllapave gjapërore është një listë e ndarë me presje e konstantave të enumeracionit.
- Variablave të tipit **enum** duhet t'u caktohen vetëm konstante të atij tipi **enum**.
- Kur deklarohet një enum, secila konstantë në deklarimin e enum-it është vlerë konstante e tipit int. Nëse nuk ia caktoni një vlerë një identifikuesi në deklarimin enum, kompajleri do ta bëjë këtë. Nëse konstanta e parë enum është me vlerë të pacaktuar, kompajleri ia jep vlerën 0. Nëse çfarëdo konstante tjetër enum është e pacaktuar, kompajleri ia jep një vlerë të barabartë me një më shumë se sa vlera e konstantes paraprake enum. Emrat e konstantave enum duhet të jenë unike, por vlerat e tyre të nënvendosura nuk duhet të jenë patjetër unike.
- Nëse duhet ta krahasosh një vlerë të thjeshtë të tipit integral me vlerën e nënvendosur të një konstanteje të enumeracionit, duhet ta përdorësh operatorin cast (të shndërrimit) për t'i bërë dy tipet të përputhen.

```

// Fig. 7.8: Craps.cs
// Klasa Craps e simulon lojën me zare: craps.
using System;

class Craps
{
    // krijoje një gjenerator të numrave të rastësishëm për ta përdorur në metodën SilliZaret
    private static Random numratERastesishem = new Random();

    // enumeracion me konstanta që e paraqesin statusin e lojës
    private enum Statusi { Vazhdim, Fituar, Humbur }

    // enumeracion me konstanta që i paraqesin hedhjet e zakonshme të zareve
    private enum EmratEZareve
    {
        SnakeEyes = 2,
        Trey = 3,
        Seven = 7,
        YoLeven = 11,
        BoxCars = 12
    }

    // e luan një lojë craps
    static void Main()
    {
        // statusiILojes mund të përmbajë Vazhdim, Fitoi apo Humbi
        Statusi statusiILojes = Statusi.Vazhdim;
        int pikaIme = 0; // pika nëse nuk ka fitore apo humbje në hedhjen e parë

        int shumaEZareve = HidhiZaret(); // hedhja e parë e zareve

        // përcaktoje statusin e lojës dhe pikën duke u bazuar në hedhjen e parë
        switch ((EmratEZareve)shumaEZareve)
        {
            case EmratEZareve.Seven: // fitore me 7 në hedhjen e parë
            case EmratEZareve.YoLeven: // fitore me 11 në hedhjen e parë
                statusiILojes = Statusi.Fituar;
                break;
            case EmratEZareve.SnakeEyes: // humbje me 2 në hedhjen e parë
            case EmratEZareve.Trey: // humbje me 3 në hedhjen e parë
            case EmratEZareve.BoxCars: // humbje me 12 në hedhjen e parë
                statusiILojes = Statusi.Humbur;
                break;
            default: // as nuk ka fituar as nuk ka humbur, pra mbaje mend pikën
                statusiILojes = Statusi.Vazhdim; // loja nuk ka mbaruar
                pikaIme = shumaEZareve; // mbaje mend pikën
                Console.WriteLine($"Pika është {pikaIme}");
                break;
        }
    }
}

```

```

// përderisa loja nuk është kryer
while (statusiILojes == Statusi.Vazhdim) // loja nuk është Fituar apo Humbur
{
    shumaEZareve = HidhiZaret(); // hidhi zaret përsëri

    // përcaktoje statusin e lojës
    if (shumaEZareve == pikaIme) // fito duke i bërë pikët
    {
        statusiILojes = Statusi.Fituar;
    }
    else
    {
        // humb duke hedhur 7 para pikëve
        if (shumaEZareve == (int)EmratEZareve.Seven)
        {
            statusiILojes = Statusi.Humbur;
        }
    }
}

// shfaqe mesazhin fituar apo humbur
if (statusiILojes == Statusi.Fituar)
{
    Console.WriteLine("Lojtari fiton");
}
else
{
    Console.WriteLine("Lojtari humb");
}

Console.ReadLine();
}

// hidhi zaret, llogarite shumën dhe shfaq rezultatat
static int HidhiZaret()
{
    // zgjedh vlera të rastësishme të zareve
    int zari1 = numratERastesishem.Next(1, 7); // hedhja e zarit të parë
    int zari2 = numratERastesishem.Next(1, 7); // hedhja e zarit të dytë

    int shuma = zari1 + zari2; // shuma e vlerave të zarit

    // shfaq rezultatat e kësaj hedhjeje
    Console.WriteLine($"Lojtari hodhi {zari1} + {zari2} = {shuma}");
    return shuma; // ktheje shumën e zareve
}
}

```

```

Lojtari hodhi 6 + 4 = 10
Pika është 10
Lojtari hodhi 6 + 2 = 8
Lojtari hodhi 2 + 1 = 3
Lojtari hodhi 4 + 4 = 8
Lojtari hodhi 3 + 2 = 5
Lojtari hodhi 1 + 3 = 4
Lojtari hodhi 6 + 2 = 8
Lojtari hodhi 6 + 6 = 12
Lojtari hodhi 6 + 4 = 10
Lojtari fiton

```

```
Lojtari hodhi 6 + 1 = 7  
Lojtari fiton
```

```
Lojtari hodhi 2 + 2 = 4  
Pika është 4  
Lojtari hodhi 3 + 4 = 7  
Lojtari humb
```

7.10 Skopi (scope, shtrirja, fushëveprimi) i deklarimeve

- Skopi i një deklarimi është pjesa e aplikacionit që mund t'i referohet entitetit të deklaruar përmes emrit të tij të pakualifikuar.
- Skopi i deklarimit të parametrut është trupi i metodës në të cilën shfaqet deklarimi.
- Skopi i ndryshores lokale është nga pika në të cilën shfaqet deklarimi deri në fund të atij blloku.
- Skopi i deklaratës së variablës-lokale që shfaqet në pjesën inicializuese të header-it (ballinës) së një urdhri for është trupi i urdhrit for dhe shprehjet e tjera në header.
- Skopi i metodës, properti-t apo fushës së klasës është i tërë trupi i klasës.
- Çdo bllok mund të përmbajë deklarime të variablave. Nëse një variabël apo parametër lokal në një metodë e ka emrin e njëjtë si një fushë, fusha fshehet deri kur blloku e përfundon ekzekutimin.

```

// Fig. 7.9: Skopi.cs
// Klasa Skopi i demonstroi skopet statike dhe lokale të variablave
using System;

class Skopi
{
    // variabël statike që është e qasshme për krejt metodat e kësaj klase
    private static int x = 1;

    // Main e krijon dhe e inicializon variablën lokale x
    // dhe i thirr metodat PerdoreVariablenLokale dhe PerdoreVariablenStatike
    static void Main()
    {
        int x = 5; // variabla lokale e metodës e fsheh variablën statike x

        Console.WriteLine($"x lokal në metodën Main është {x}");

        // PerdoreVariablenLokale e ka x-in e vet lokal
        PerdoreVariablenLokale();

        // PerdoreVariablenStatike e përdor variablën statike x të klasës Skopi
        PerdoreVariablenStatike();

        // PerdoreVariablenLokale e ri-inicializon x-in e vet lokal
        PerdoreVariablenLokale();

        // variabla statike x e klasës Skopi e ruan vlerën e vet
        PerdoreVariablenStatike();

        Console.WriteLine($"\\nx lokal në metodën Main është {x}");

        Console.ReadLine();
    }

    // krijoje dhe inicializoj variablën lokale x gjatë secilës thirrje
    static void PerdoreVariablenLokale()
    {
        int x = 25; // inicializohet sa herë të thirret PerdoreVariablenLokale

        Console.WriteLine(
            $"\\nx lokal gjatë hyrjes në metodën PerdoreVariablenLokale është {x}");
        ++x; // e ndryshon variablën lokale x të kësaj metode
        Console.WriteLine(
            $"x lokal para daljes nga metoda PerdoreVariablenLokale është {x}");
    }

    // ndryshoj variablën statike x të klasës Skopi gjatë secilës thirrje
    static void PerdoreVariablenStatike()
    {
        Console.WriteLine("\\nvariabla statike x gjatë hyrjes në metodën " +
            $"PerdoreVariablenStatike është {x}");
        x *= 10; // e ndryshon variablën statike x të klasës Skopi
        Console.WriteLine("variabla statike x para daljes " +
            $"nga metoda PerdoreVariablenStatike është {x}");
    }
}

```



```

x lokal në metodën Main është 5

x lokal gjatë hyrjes në metodën PerdoreVariablenLokale është 25
x lokal para daljes nga metoda PerdoreVariablenLokale është 26

variabla statike x gjatë hyrjes në metodën PerdoreVariablenStatike është 1
variabla statike x para daljes nga metoda PerdoreVariablenStatike është 10

x lokal gjatë hyrjes në metodën PerdoreVariablenLokale është 25
x lokal para daljes nga metoda PerdoreVariablenLokale është 26

variabla statike x gjatë hyrjes në metodën PerdoreVariablenStatike është 10
variabla statike x para daljes nga metoda PerdoreVariablenStatike është 100

x lokal në metodën Main është 5

```

7.11 Steku i thirrjes së metodave (Method-Call Stack) dhe regjistrat e aktivizimit (activation records)

- Steku njihet si strukturë e shënimeve LIFO (last-in, first-out, pra ai që hyn i fundit, del i pari) – elementi i fundit që futet (pushed, inserted) në stek është i pari që nxirret (pop) nga steku.
- Steku i thirrjes së metodave (nganjëherë i quajtur steku i ekzekutimit të programit – program-execution stack) është strukturë e shënimeve (data structure) që e mbështet mekanizmin e thirrjes/kthimit të metodave. Ai e mbështet edhe krijimin, mirëmbajtjen dhe shkatërrimin e variablave lokale të secilës metodë të thirrur.
- **(7.11.2 Pjesët e Stekut - Stack Frames)** Secila metodë në fund duhet t’ia kthejë kontrollin metodës që e ka thirrur atë. Steku i thirrjes së metodave është struktura e përsosur e shënimeve për trajtimin e këtij informacioni.
- Sa herë që një metodë e thirr një metodë tjetër, një informacion (i quajtur **stack frame**, pjesë e stekut) futet në stek. Ky informacion e përmban adresën e kthimit që i duhet metodës së thirrur për t’u kthyer te metoda thirrëse.
- Nëse një metodë e thirrur kthehet në vend se ta thërrasë një metodë tjetër para kthimit, pjesa e stekut për thirrjen e metodës hiqet nga steku, dhe kontrolli transferohet tek adresa kthyesë në pjesën e nxjerrë të stekut.
- Të njëjtat teknika zbatohen kur një metodë i qaset një properti-e apo kur një properti e thirr një metodë.

- **(7.11.3 Variablat lokale dhe Pjesët e stekut)** Pjesët e stekut (stack frames) e përmbajnë edhe memorien për parametrat dhe çdo variabël lokale që e deklaron një metodë.
- Kur një metodë kthehet (me return) – dhe më nuk i duhen variablat e saj lokale – pjesa e saj e stekut nxirret nga steku, dhe ato variabla lokale nuk ekzistojnë më.
- **(7.11.4 Stack Overflow – Tejmbushja-Vërshimi i Stekut)** Nëse ndodhin më shumë thirrje të metodave se sa që mund të ruhen shënimet e tyre të aktivizimit në stekun e thirrjes së metodave, ndodh një gabim fatal i njohur si stack overflow (tejmbushja-vërshimi i stekut).

```
// Fig. 7.10: TestIKatrorit.cs
// Metoda Katrorizoje e përdorur për ta demonstruar
// stekun e thirrjeve të metodës (call stack)
// dhe regjistrat e aktivizimit (activation records)
using System;

class TestIKatrorit
{
    static void Main()
    {
        int x = 10; // vlera për ta katrorizuar (variabël lokale në main)
        Console.WriteLine($"x i katrorizuar: {Katrorizoje(x)}");
        Console.ReadKey();
    }

    // e kthen katrorin e një numri të plotë
    static int Katrorizoje(int y) // y është variabël lokale
    {
        return y * y; // llogarite katrorin e y dhe ktheje rezultatin
    }
}
```

```
x i katrorizuar: 100
```

Step 1: Operating system calls `Main` to begin program execution

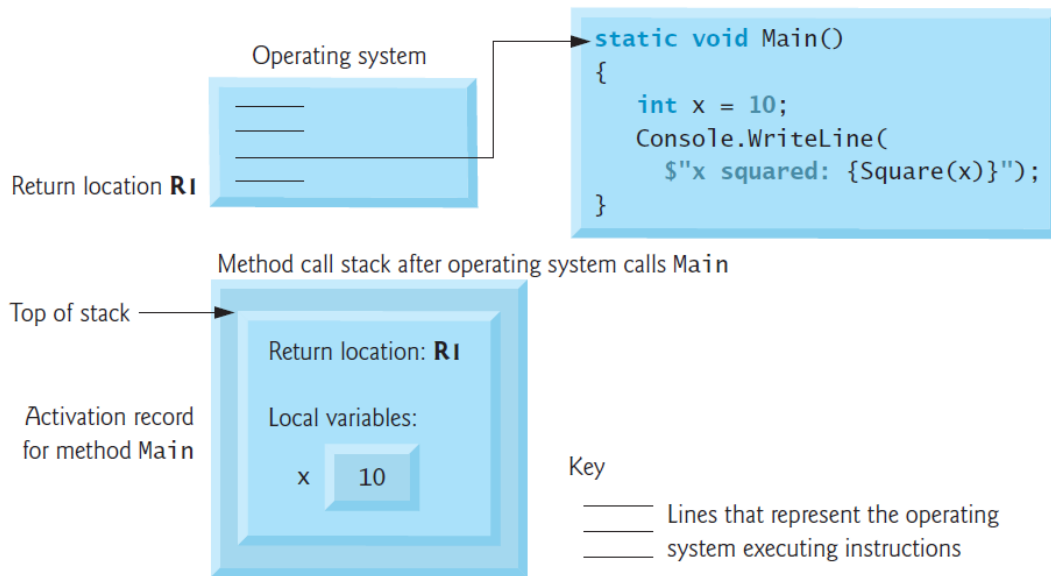


Fig. 7.11 | Method-call stack after the operating system calls `main` to execute the program.

Step 2: Main calls method Square to perform calculation

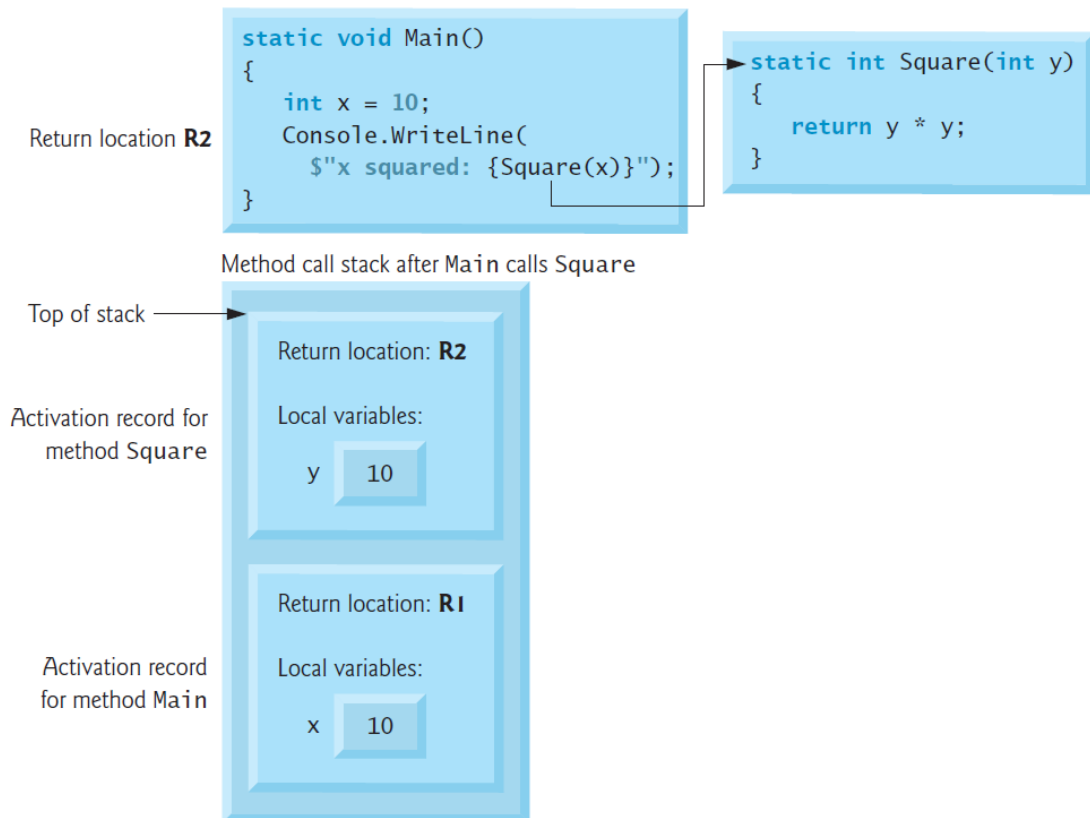


Fig. 7.12 | Method-call stack after `Main` calls `square` to perform the calculation.

7.12 Mbingarkimi i metodës (method overloading)

- **(7.12.1 Deklarimi i metodave të mbingarkuara)** Metodatat me emrin e njëjtë mund të deklarohen në klasën e njëjtë, përderisa kanë grupe (sete) të ndryshme të parametrave. Kjo quhet mbingarkim i metodës. Kur thirret një metodë e mbingarkuar, kompajleri i C# e zgjedh metodën e duhur duke e ekzaminuar numrin, tipet dhe radhën e argumenteve në thirrje.

```
// Fig. 7.14: MbingarkimiIMetodes.cs
// Deklarimet e metodës së mbingarkuar.
using System;

class MbingarkimiIMetodes
{
    // testoji metodat e mbingarkuara Katrori

    static void Main()
    {
        Console.WriteLine($"Katrori i numrit të plotë 7 është {Katrori(7)}");
        Console.WriteLine($"Katrori i numrit double 7.5 është {Katrori(7.5)}");
        Console.ReadLine();
    }

    // metoda Katrori me argument int
    static int Katrori(int vleraInt)
    {
        Console.WriteLine(
            $"E thirra metodën Katrori me argumentin int: {vleraInt}");
        return vleraInt * vleraInt;
    }

    // metoda Katrori me argument double
    static double Katrori(double vleraDouble)
    {
        Console.WriteLine(
            $"E thirra metodën Katrori me argumentin double: {vleraDouble}");
        return vleraDouble * vleraDouble;
    }
}
```

```
E thirra metodën Katrori me argumentin int: 7
Katrori i numrit të plotë 7 është 49
E thirra metodën Katrori me argumentin double: 7.5
Katrori i numrit double 7.5 është 56.25
```

- **(7.12.2 Si bëhet dallimi ndërmjet metodave të mbingarkuara)** Kompajleri i dallon metodat e mbingarkuara sipas nënshkrimit (signature) të tyre. Nënshkrimi është kombinim i emrit të metodës dhe numrit, tipeve dhe radhës së parametrave të saj. Nënshkrimi e përfshin edhe mënyrën se si pasohen (dërgohen) ata parametra, që mund të ndryshohet me fjalët kyçe **ref** dhe **out**.
- **(7.12.3 Tipet kthyesë të metodave të mbingarkuara)** Kompajleri do ta gjenerojë (prodhojë) një gabim kur dy metoda e kanë nënshkrimin e njëjtë, por tipe të ndryshme të kthimit. Metodatat e mbingarkuara mund të kenë tipe të njëjta apo të ndryshme nëse metodat kanë lista të ndryshme të parametrave.

7.13 Parametrat opsionale

```
// Fig. 7.15: LlogaritiFuqite.cs
// Demonstrimi i parametrave opsional me metodën Fuqia
using System;

class LlogaritiFuqite
{
    // thirrje metodën Fuqia me dhe pa argumente opsionale
    static void Main()
    {
        Console.WriteLine($"Fuqia(10) = {Fuqia(10)}");
        Console.WriteLine($"Fuqia(2, 10) = {Fuqia(2, 10)}");
        Console.ReadLine();
    }

    // përdore iterimin për ta llogaritur fuqinë
    static int Fuqia(int vleraEBazes, int vleraEEksponentit = 2)
    {
        int rezultati = 1;

        for (int i = 1; i <= vleraEEksponentit; ++i)
        {
            rezultati *= vleraEBazes;
        }

        return rezultati;
    }
}
```

```
Fuqia(10) = 100
Fuqia(2, 10) = 1024
```

- Metodatat mund të kenë parametra opsionalë, të cilët e lejojnë metodën thirrëse ta ndryshojë numrin e argumenteve që pasohen (dërgohen). Një parametër opsional e specifikon një vlerë të paracaktuar (default) që i caktohet parametrut nuk përfshihet argumenti opsional.
- Metodatat mund ta kenë një apo më shumë parametra opsionalë. Të gjithë parametrat opsionalë duhet të vendosen djathtas (pra pas) parametrave jo-opsionalë të metodës.
- Kur një parametër e ka një vlerë të paracaktuar (default), thirrësi e ka opsionin e kalimit (lënies zbrazët) të atij argumenti.

7.14 Parametrat e emërtuar (named parameters)

- Normalisht, kur thirret një metodë, vlerat e argumenteve – me radhë – u caktohen parametrave nga e majta në të djathtë në listën e parametrave.
- C# e ofron një veçori të quajtur parametrat e emërtuar, që ju mundësojnë t'i thirrni metodat që pranojnë parametra opsionalë duke i ofruar vetëm argumentet opsionale që dëshironi t'i specifikoni. Për ta bërë këtë, e specifikoni në mënyrë eksplicite emrin dhe vlerën e parametrut – të ndarë me një dypikësh (:) – në listën e argumenteve të thirrjes së metodës.

Consider the following Cube method:

```
static int Cube(int x)
{
    return x * x * x;
}
```

In C# 6, this can be expressed with an **expression-bodied method** as

```
static int Cube(int x) => x * x * x;
```

```
public bool IsNoFaultState =>
    State == "MA" || State == "NJ" || State == "NY" || State == "PA";
```

7.16 Rekursioni

- Një metodë rekursive e thërret veten, ose direkt (drejtpërdrejt) ose indirekt (tërthorazi) përmes një metode tjetër.

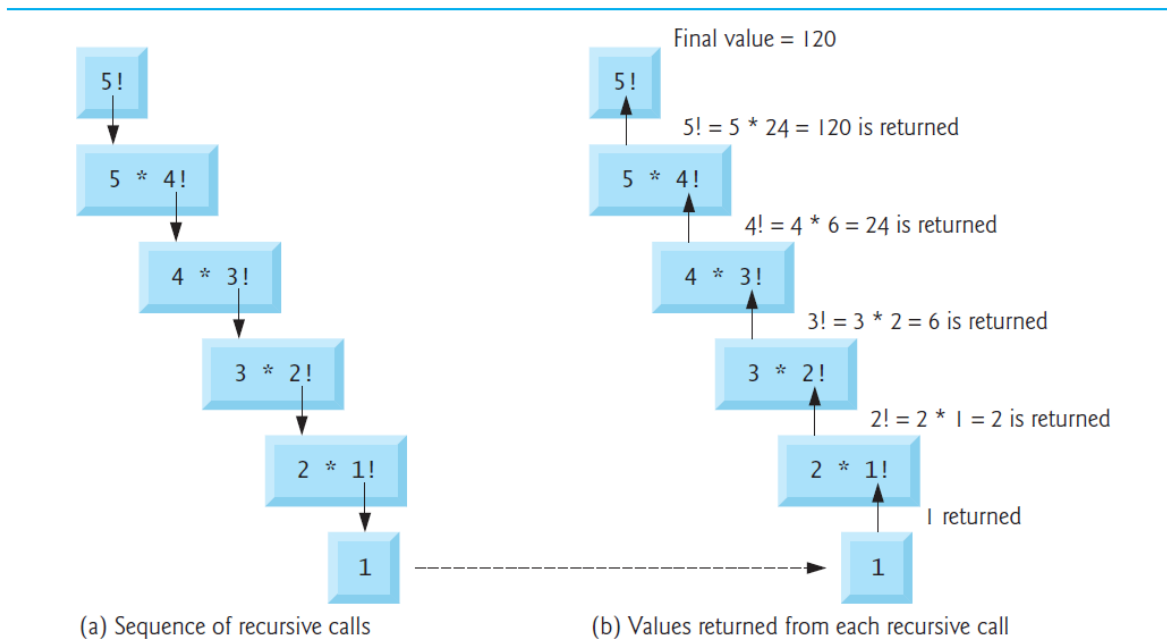


Fig. 7.16 | Recursive evaluation of 5!.

7.16.1 Rastet bazë dhe thirrjet rekursive

- Kur thirret një metodë rekursive për ta zgjidhur një problem, metoda në të vërtetë është e aftë ta zgjidhë vetëm rastin (rastet) më të thjeshtë (më të thjeshta) apo rastin/rastet bazë. Nëse metoda thirret me rast bazë, metoda e kthen një rezultat.
- Nëse metoda thirret me një problem më kompleks, metoda e ndan problemin në dy pjesë konceptuale: një pjesë të cilën metoda di se si të veprojë dhe një pjesë për të cilën nuk di se si të veprojë. Meqë ky problem i ri duket si problemi fillestar (origjinal), metoda e thërret një kopje të freskët të vetes për të punuar në problemin më të vogël; kjo procedurë quhet (referohet) si thirrje rekursive dhe quhet edhe hapi i rekursionit.

7.16.2 Llogaritjet rekursive të faktorielit

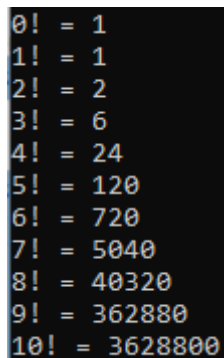
- Një deklarim rekursiv i metodës faktoriel arrihet duke e studiuar relacionin:

$$n! = n \cdot (n - 1)!$$

```
// Fig. 7.17: TestIFaktorielit.cs
// Metodë rekursive e faktorielit.
using System;

class TestIFaktorielit
{
    static void Main()
    {
        // llogariti faktorielët e numrave 0 deri në 10
        for (long numeruesi = 0; numeruesi <= 10; ++numeruesi)
        {
            Console.WriteLine($"{numeruesi}! = {Faktorieli(numeruesi)}");
        }
        Console.ReadLine();
    }

    // deklarimi rekursiv i metodës Faktorieli
    static long Faktorieli(long numri)
    {
        // rasti bazë
        if (numri <= 1)
        {
            return 1;
        }
        else // hapi i rekursionit (hapi rekursiv)
        {
            return numri * Faktorieli(numri - 1);
        }
    }
}
```



```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

7.17 Tipet vlerë vs Tipet referencë

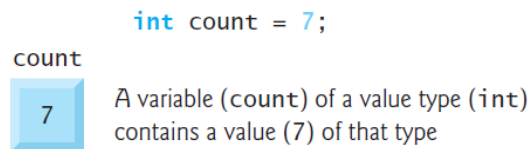


Fig. 7.18 | Value-type variable.

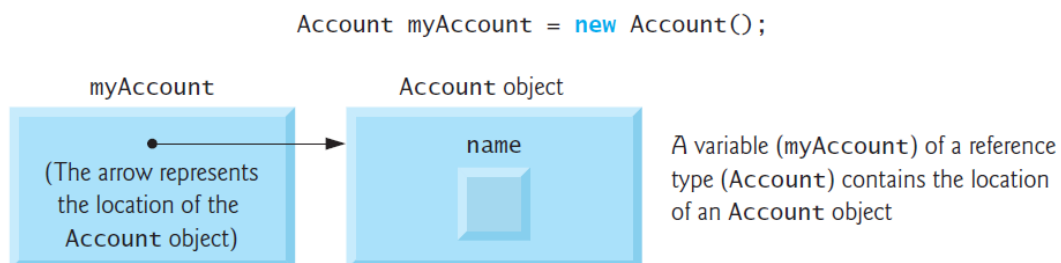


Fig. 7.19 | Reference-type variable.

- Tipet në C# ndahen në tipe vlerë (value types) dhe tipe referencë (reference types).
- Tipet e thjeshta të C# (si **int**), tipet **enum** dhe tipet **struct** janë të gjitha tipe vlerë.
- Një variabël e një tipi vlerë thjesht e përmban një vlerë të atij tipi.
- Një variabël e tipit referencë (nganjëherë e quajtur referencë) e përmban lokacionin (vendndodhjen) ku ruhen shënimet e referuara nga ajo variabël. Një variabël e tillë thuhet se i referohet një objekti.
- Variablat e instancës të tipit referencë inicializohen si parazgjedhje (default) si **null**.
- Tipi **string** është tip referencë. Një variabël string me vlerën **null** nuk është string bosh, i cili paraqitet me "" apo **string.Empty**.
- Vlera **null** e paraqet një referencë që nuk i referohet një objekti.
- **string**-u bosh është objekt **string** që nuk e përmban asnjë karakter.

7.18 Pasimi (dërgimi) i argumenteve me vlerë (by value) dhe me referencë (by ref.)

- Dy mënyra për t'ua pasuar (dërguar) argumentet metodave në shumë gjuhë programuese janë pass-by-value (pasimi me vlerë) dhe pass-by-reference (pasimi me referencë).
- Kur një argument pasohet me vlerë (opsioni i paracaktuar, default), metodës së thirrur i pasohet një *kopje* e vlerës së argumentit. Ndryshimet në kopje nuk ndikojnë në vlerën e ndryshores origjinale të thirrësi.
- Kur një argument pasohet me referencë, thirrësi ia jep metodës mundësinë për t'iu qasur dhe për t'i ndryshuar drejtpërdrejt shënimet e thirrësit.
- Variablat me tip vlerë ruajnë vlera, kështu që specifikimi i një variable me tip vlerë në një thirrje të metodës ia pason metodës një kopje të vlerës së asaj variable. Variablat e tipit referencë ruajnë referenca në objekte, prandaj specifikimi i një variable të tipit referencë si argument ia pason metodës një kopje të referencës aktuale që i referohet objektit.
- Kur kthehet informacion nga një metodë përmes urdhrit **return**, metoda e kthen një kopje të vlerës së ruajtur në një variabël të tipit vlerë apo një kopje të referencës së ruajtur në një variabël të tipit referencë.

7.18.1 Parametrat ref dhe out

- C# i ofron fjalët kyçe (keywords) **ref** dhe **out** për t'i pasuar variablat me referencë.
- Një parametër **ref** tregon se një argument do t'i pasohet metodës me referencë – metoda e thirrur do të ketë mundësi ta ndryshojë variablën origjinale që është te thirrësi.
- Një parametër **out** tregon se një variabël ndoshta e painicializuar do t'i pasohet metodës me referencë dhe se metoda e thirrur do t'ia caktojë një vlerë variablës origjinale që është te thirrësi.
- Një metodë mund t'ia kthejë vetëm një vlerë thirrësit të vet përmes një urdhri return, por mund të kthejë shumë vlera duke specifikuar shumë parametra dalës-output (**ref** dhe/apo **out**).

- Kur një variabël i pasohet një metode me parametër referencë, duhet ta paraprintë variablën me fjalën e njëjtë kyçe (**ref** apo **out**) që është përdorur për ta deklaruar parametrin referencë.

```
// Fig. 7.20: ParametratReferenceDheOutput.cs
// Parametrat referencë, output (dalës) dhe value (vlerë).
using System;
class ParametratReferenceDheOutput
{
    // thirri metodat me parametra referencë, output (dalës) dhe value (vlerë)
    static void Main()
    {
        int y = 5; // inicializojë y me 5
        int z; // e deklaron z, por nuk e inicializon

        // shfaq vlerat fillestare (origjinale) të y dhe z
        Console.WriteLine($"Vlera fillestare (origjinale) e y: {y}");
        Console.WriteLine("Vlera fillestare e z: e painicializuar\n");

        // pasoji y dhe z me referencë (by reference)
        KatrorizojëRef(ref y); // duhet të përdoret fjala kyçe ref
        KatrorizojëOut(out z); // duhet të përdoret fjala kyçe out

        // shfaq vlerat e y dhe z pasi janë ndryshuar nga
        // metodat KatrorizojëRef dhe KatrorizojëOut, përkatësisht
        Console.WriteLine($"Vlera e y pas KatrorizojëRef: {y}");
        Console.WriteLine($"Vlera e z pas KatrorizojëOut: {z}\n");

        // pasoji y dhe z me vlerë (by value)
        Katrorizojë(y);
        Katrorizojë(z);

        // shfaq vlerat e y dhe z pasi i janë pasuar metodës Katrorizojë
        // për të demonstruar se argumentet e pasuara me vlerë nuk ndryshohen
        Console.WriteLine($"Vlera e y pas thirrjes Katrorizojë: {y}");
        Console.WriteLine($"Vlera e z pas thirrjes Katrorizojë: {z}");
        Console.ReadLine();
    }

    // e përdor parametrin referencë x për ta ndryshuar variablën e thirrësit
    static void KatrorizojëRef(ref int x)
    {
        x = x * x; // e katrorizon vlerën e variablës së thirrësit
    }

    // e përdor parametrin dalës (output) për t'ia caktuar një vlerë
    // një variable të painicializuar
    static void KatrorizojëOut(out int x)
    {
        x = 6; // ia cakton një vlerë variablës së thirrësit
        x = x * x; // e katrorizon vlerën e variablës së thirrësit
    }

    // parametri x e pranon një kopje të vlerës së pasuar si argument,
    // kështu që kjo metodë nuk mund ta ndryshojë variablën e thirrësit
    static void Katrorizojë(int x)
    {
        x = x * x;
    }
}
```

```
Vlera fillestare (origjinale) e y: 5  
Vlera fillestare e z: e painicializuar
```

```
Vlera e y pas KatrorizojeRef: 25  
Vlera e z pas KatrorizojeOut: 36
```

```
Vlera e y pas thirrjes Katrorizoje: 25  
Vlera e z pas thirrjes Katrorizoje: 36  
_
```

8. Vargjet (Arrays); Hyrje në trajtimin e përjashtimeve (Exception handling)

8.1 Hyrje

- Vargjet janë struktura të shënimeve (struktura të të dhënave, data structures) që përbëhen nga elemente të shënimeve të ndërlidhura të tipit të njëjtë. Vargjet janë entitete me gjatësi fikse – ato mbeten me gjatësinë e njëjtë pasi të krijohen.

8.2 Vargjet

- Vargjet janë tipe referencë.
- Elementet e një vargu mund të jenë ose tipe vlerë ose tipe referencë (duke përfshirë vargje të tjera).
- Një aplikacion i referohet një elementi të vargut me shprehje për qasje në varg që e përfshin emrin e vargut, të pasuar nga indeksi i elementit të caktuar në kllapa të mesme (katrore: []).
- Elementi i parë në çdo varg e ka indeksin zero dhe nganjëherë quhet elementi i zero-të (zeroth element).
- Properti **Length** e vargut e kthen numrin e elementeve në varg.

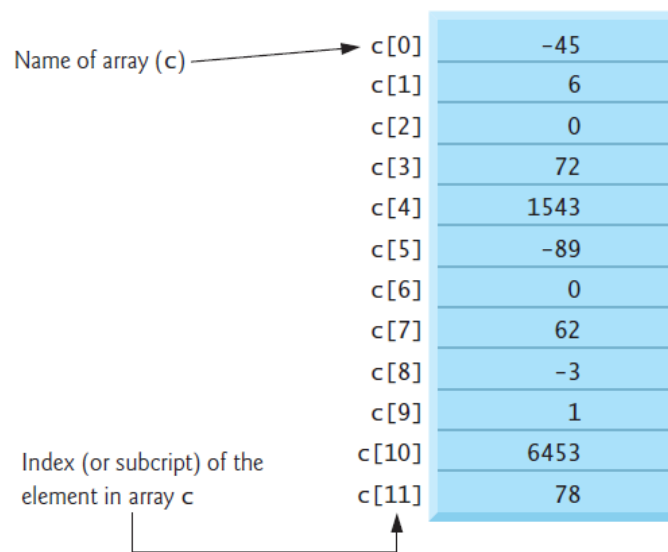


Fig. 8.1 | A 12-element array.

8.3 Deklarimi dhe krijimi i vargjeve

- Për ta krijuar një instancë të vargut, e specifikoni tipin dhe numrin e elementeve të vargut si pjesë e shprehjes së krijimit të vargut që e përdor fjalën kyçe **new**, si:

```
int[] a = new int[12];
```

- Kur krijohet një varg, secili elementi i vargut merr vlerë të paracaktuar (default).
- Çdo element i një vargu me tip-vlerë e përmban një vlerë të tipit të deklaruar të vargut. Në një varg të tipit referencë, çdo element është referencë të një objekt i tipit të deklaruar të vargut apo **null**.

8.4 Shembuj që përdorin vargje

```
// Fig. 8.2: InicojeVargun.cs
// Krijimi i një vargu.
using System;

class InicojeVargun
{
    static void Main()
    {
        // krijoje hapësirën për vargun
        // dhe inicializojë me zero të paracaktuara (default)
        int[] vargu = new int[5]; // vargu i përmban 5 elemente int

        Console.WriteLine($"{ "Indeksi" } { "Vlera", 8 }"); // titujt

        // afishojë (nxirre në dalje) vlerën e secilit element të vargut
        for (int numeruesi = 0; numeruesi < vargu.Length; ++numeruesi)
        {
            Console.WriteLine($"{ numeruesi, 7 } { vargu[numeruesi], 8 }");
        }
        Console.Read();
    }
}
```

Indeksi	Vlera
0	0
1	0
2	0
3	0
4	0

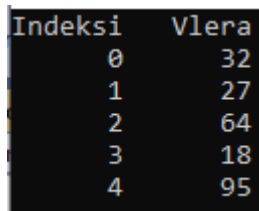
- **(8.4.2 Përdorimi i inicializuesit të vargut)** Një aplikacion mund ta krijojë një varg dhe t'i inicializojë elementet e tij me inicializues të vargut, që është listë e ndarë me presje (comma-separated) e shprehjeve (e quajtur listë inicializuese) e mbyllur në kllapa gjarpërore.


```
// Fig. 8.3: InicojeVargun.cs
// Inicializimi i elementeve të një vargu me inicializues të vargut.
using System;

class InicojeVargun
{
    static void Main()
    {
        // lista inicializuese e specifikon vlerën e secilit element
        int[] vargu = { 32, 27, 64, 18, 95 };

        Console.WriteLine($"{ "Indeksi" } { "Vlera", 8 }"); // titujt

        // afishoje (nxirre në dalje) vlerën e secilit element të vargut
        for (int numeruesi = 0; numeruesi < vargu.Length; ++numeruesi)
        {
            Console.WriteLine($"{ numeruesi, 7 } { vargu[numeruesi], 8 }");
        }
        Console.Read();
    }
}
```



Indeksi	Vlera
0	32
1	27
2	64
3	18
4	95

- **(8.4.3 Llogaritja e një vlere për ta ruajtur në çdo element të vargut)** Konstantet duhet të inicializohen kur të deklarohen dhe nuk mund të ndryshohen më pas.

```
// Fig. 8.4: InitArray.cs
// Llogaritja e vlerave që do të vendosen në elementet e një vargu.
using System;

class InicojeVargun
{
    static void Main()
    {
        const int GjatesiaEVargut = 5; // krijoje një konstantë të emëruar
        int[] vargu = new int[GjatesiaEVargut]; // krijoje vargun

        // llogarite vlerën për secilin element të vargut
        for (int numeruesi = 0; numeruesi < vargu.Length; ++numeruesi)
        {
            vargu[numeruesi] = 2 + 2 * numeruesi;
        }

        Console.WriteLine($"{ "Indeksi" } { "Vlera", 8 }"); // titujt

        // afishoje (nxirre në dalje) vlerën e secilit element të vargut
        for (int numeruesi = 0; numeruesi < vargu.Length; ++numeruesi)
        {
            Console.WriteLine($"{ numeruesi, 7 } { vargu[numeruesi], 8 }");
        }
        Console.Read();
    }
}
```

Indeksi	Vlera
0	2
1	4
2	6
3	8
4	10

8.4.4 Mbledhja si shumë e elementeve të një vargu

```
// Fig. 8.5: MbledheVargun.cs
// Llogaritja e shumës së elementeve të një vargu
using System;

class MbledheVargun
{
    static void Main()
    {
        int[] vargu = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
        int totali = 0;

        // shtoja totalit vlerën e secilit element
        for (int numeruesi = 0; numeruesi < vargu.Length; ++numeruesi)
        {
            totali += vargu[numeruesi];
        }

        Console.WriteLine($"Totali i elementeve të vargut: {totali}");
        Console.Read();
    }
}
```

```
Totali i elementeve të vargut: 849
```

8.4.5 Iterimi nëpër vargje me *foreach*

- Urdhri (formulimi) **foreach** iteron nëpër elementet e një vargu apo koleksioni të tërë. Sintaksa e urdhrit **foreach** është:

```
foreach (tipi identifikuesi in emriVargut)
{
    urdhri
}
```

ku *tipi* dhe *identifikuesi* janë tipi dhe emri i variablës iteruese, dhe *emriVargut* është vargu nëpër të cilin iterohet.

- Ballina e (header-i i) **foreach** mund të lexohet në mënyrë koncize (të ngjeshur, të shkurtër) si “për secilin iterim (për secilën përsëritje), caktoja elementin e radhës të vargut – variablës së iterimit, pastaj ekzekutoje urdhrin vijues.”

- Variabla e iterimit e urdhrit foreach mund të përdoret vetëm për t'iu qasur elementeve të vargut, jo për t'i ndryshuar ato. Çfarëdo tentimi për ta ndryshuar vlerën e ndryshores së iterimit në trupin e një urdhri foreach do të shkaktojë gabim të kompajlimit.

```
// Fig. 8.6: TestForEach.cs
// Përdorimi i urdhrit foreach për t'i mbledhur (totalizuar)
// numrat e plotë në një varg
using System;

class TestForEach
{
    static void Main()
    {
        int[] vargu = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
        int totali = 0;

        // shtoja totalit vlerën e secilit element
        foreach (int numri in vargu)
        {
            totali += numri;
        }

        Console.WriteLine($"Totali i elementeve të vargut: {totali}");
        Console.Read();
    }
}
```

```
Totali i elementeve të vargut: 849
```

8.4.6 Përdorimi i bar charts (grafikëve me shirita) për t'i shfaqur grafikisht shënimet e vargut; type inference (nxjerrja e tipit) me *var*

- Në një element të formatit (format item), specifikuesi i formatit **D** tregon se vlera duhet të formatohet si integer (numër i plotë), dhe numri pas **D**-së tregon se sa shifra duhet t'i përmbajë ky numër i plotë i formatuar.
- Fjala kyçe **var** e deklaron një variabël lokale dhe e lejon kompajlerin ta përcaktojë tipin e variablës, bazuar në inicializuesin e variablës. Ky proces njihet si type inference (nxjerrje e tipit), dhe variablat lokale të deklaruara në këtë mënyrë njihen si variabla lokale të tipuara në mënyrë implicite (implicitly typed local variables).
- Fjala kyçe **var** nuk mund të përdoret për ta deklaruar një varg që është inicializuar përmes një liste të inicializuesit – inicializuesit e tillë mund të përdoren edhe me koleksione, kështu që kompajleri nuk mund ta nxjerrë informatën se a duhet të jetë variabla varg apo koleksion.

```
// Fig. 8.7: BarChart.cs
// Aplikacion që shfaq bar chart (grafik me shirita)
using System;

class BarChart
{
    static void Main()
    {
        int[] vargu = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 }; // shpërndarja

        Console.WriteLine("Shpërndarja e notave:");

        // për secilin element të vargut,
        // afishojë një bar (shirit) të chart-it (grafikut)
        for (var numeruesi = 0; numeruesi < vargu.Length; ++numeruesi)
        {
            // afishoji bar labels (etiketat e shiritave)
            // ("00-09: ", ..., "90-99: ", "100: ")
            if (numeruesi == 10)
            {
                Console.Write(" 100: ");
            }
            else
            {
                Console.Write($"{numeruesi * 10:D2}-{numeruesi * 10 + 9:D2}: ");
            }

            // shfaqë bar-in (shiritin) e yjeve (asterisks)
            for (var yjet = 0; yjet < vargu[numeruesi]; ++yjet)
            {
                Console.Write("*");
            }

            Console.WriteLine(); // nisë një rresht të ri të daljes (output-it)
        }

        Console.Read();
    }
}
```

```
Shpërndarja e notave:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
100: *
```

8.4.7 Përdorimi i elementeve të një vargu si numërues

```
// Fig. 8.8: SilleZarin.cs
// Sille (rrokullise) një zar gjashtë-faqësh 60,000,000 herë.
using System;

class SilleZarin
{
    static void Main()
    {
        var numratERastesishem = new Random(); // gjenerator i numrave të rastësishëm
        var frekuenca = new int[7]; // varg i numëruesve të frekuencës (shpeshtësisë)

        // sille (rrokullise) zarin 60,000,000 herë;
        // përdore vlerën e zarit si indeks të frekuencës
        for (var rrokullisja = 1; rrokullisja <= 60000000; ++rrokullisja)
        {
            ++frekuenca[numratERastesishem.Next(1, 7)];
        }

        Console.WriteLine($"{ "Faqja" } { "Frekuenca", 10 }");

        // afishoje (qite në dalje) vlerën e secilit element të vargut
        for (var faqja = 1; faqja < frekuenca.Length; ++faqja)
        {
            Console.WriteLine($"{ faqja, 5 } { frekuenca[faqja], 10 }");
        }
        Console.Read();
    }
}
```

```
Faqja Frekuenca
1 9998435
2 10000884
3 9999495
4 9994943
5 10004785
6 10001458
```

8.5 Përdorimi i vargjeve për t'i analizuar rezultatet e sondazhit; Hyrje në trajtimin e përjashtimeve (exception handling)

```
// Fig. 8.9: SondazhiIStudenteve.cs
// Aplikacion për analizë të sondazhit.
using System;

class SondazhiIStudenteve
{
    static void Main()
    {
        // vargu i përgjigjeve të studentëve
        // (zakonisht të dhënat futen gjatë kohës së ekzekutimit)
        int[] pergjigjet = {1, 2, 5, 4, 3, 5, 2, 1, 3, 3, 1, 4, 3, 3, 3,
            2, 12, 3, 3, 2, 14};
        var frekuenca = new int[6]; // varg i numëruesve të frekuencave

        // për secilën përgjigje, zgjedhe elementin e përgjigjeve
        // dhe përdore atë vlerë si indeks të frekuencës
        // për ta përcaktuar elementin që do të rritet
        for (var pergjigjja = 0; pergjigjja < pergjigjet.Length; ++pergjigjja)
        {
            try
            {
                ++frekuenca[pergjigjet[pergjigjja]];
            }
            catch (IndexOutOfRangeException ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(
                    $"    pergjigjet[{pergjigjja}] = {pergjigjet[pergjigjja]}\n");
            }
        }

        Console.WriteLine($"{"Vlerësimi"}{"Frekuenca",10}");

        // afishoje (qite në dalje) vlerën e secilit element të vargut
        for (var ratingu = 1; ratingu < frekuenca.Length; ++ratingu)
        {
            Console.WriteLine($"{"ratingu",9}{frekuenca[ratingu],10}");
        }
        Console.Read();
    }
}
```

```
Index was outside the bounds of the array.
    pergjigjet[16] = 12

Index was outside the bounds of the array.
    pergjigjet[20] = 14

Vlerësimi Frekuenca
    1         3
    2         4
    3         8
    4         2
    5         2
```

8.5.2 Trajtimi i përjashtimeve: Procesimi i përgjigjes së parregullt

- Përjashtimi e tregon një problem që ndodh gjersa ekzekutohet programi.
- Trajtimi i përjashtimeve ju mundëson të krijoni programe që janë tolerante ndaj gabimeve të cilat mund t'i zgjidhin përjashtimet.

8.5.3 Urdhri *try*

- Për ta trajtuar një përjashtim, vendoseni çdo kod që mund ta hedhë (gjuajë) ndonjë përjashtim në urdhër **try**.
- Blloku **try** e përmban kodin që mund ta hedhë një përjashtim, dhe blloku **catch** e përmban kodin që e trajton përjashtimin nëse ndodh një i tillë.
- Mund të keni shumë blloqe **catch** për të trajtuar tipe të ndryshme të përjashtimeve që mund të hidhen në bllokun përkatës **try**.
- Kur përfundon një bllok **try**, çdo variabël e deklaruar në bllokun **try** është jashtë skopit (fushëveprimit).

8.5.3 Ekzekutimi i bllokut *catch*

- Një bllok **catch** e deklaron një tip dhe një parametër të përjashtimit. Brenda bllokut **catch**, mund ta përdorni identifikuesin e parametrin për të ndërvepruar me një objekt të zënë të përjashtimit.
- Kur ekzekutohet një program, indeksat e elementit të vargut kontrollohen për validitet (vlefshmëri) – të gjithë indeksat duhet të jenë më të mëdhenj apo të barabartë me 0 dhe më të vogël se sa gjatësia e vargut. Nëse tentohet të përdoret një indeks jovalid për t'iu qasur një elementi, ndodh një përjashtim **IndexOutOfRangeException**.

8.5.5 Properti *Message* e parametrin të përjashtimit

- Properti Message e një objekti të përjashtimit e kthen mesazhin e gabimit të përjashtimit.

8.6 Studim i rastit: Simulimi i përzierjes dhe (shpër)ndarjes së letrave të lojës (kartave)

8.6.1 Klasa Leter dhe Properti-t Getter-Only (Vetëm Marrëse) të implementuara automatikisht (auto-implemented)

- Para C# 6 properti-t e implementuara automatikisht kërkonin edhe aksesor (qasës) **get** edhe **set**.
- C# 6 i përkrah properti-t e implementuara automatikisht që janë read only (vetëm për lexim).
- Properti-t që janë getter-only (vetëm marrëse) duhet të inicializohen ose në deklarimet e tyre ose në të gjithë konstruktorët e tipit.
- Inicializimi i një properti-e të implementuar automatikisht në deklarin e saj është një veçori tjetër e C# 6 e njohur si auto-property initializers (inicializuesit e propertive automatike).
- Metoda **ToString** e një objekti thirret në mënyrë implicite në shumë raste kur objekti përdoret aty ku pritët një **string**, siç është ngjitja e një objekti me një **string**.

```
// Fig. 8.10: Leter.cs
// Klasa Leter e paraqet një letër të lojës (kartë)
class Leter
{
    private string Vlera { get; } // Vlera e letrës ("Pikë", "Dysh", ...)
    private string Grupi { get; } // Grupi i letrës: "Zemër" (Hearts), "Diamant" (Diamonds)
                                   // "Rrush" (Clubs), "Gjeth" (Spades)

    // konstruktori me dy parametra e inicializon Vlerën dhe Grupin e letrës
    public Leter(string vlera, string grupi)
    {
        Vlera = vlera; // inicializojë vlerën e letrës
        Grupi = grupi; // inicializojë ngjyrën e letrës
    }

    // kthejë paraqitjen string të objektit Leter
    public override string ToString() => $"{Vlera} - {Grupi}";
}

```

```
// Fig. 8.11: PakoELerave.cs
// Klasa PakoELerave e përfaqëson një pako të letrave të lojës.
using System;

class PakoELerave
{
    // krijojë një objekt Random për ta bashkëpërdorur ndërmjet objekteve PakoELerave
    private static Random numratERastesishem = new Random();

    private const int NumriILerave = 52; // numri i letrave në një pako
    private Leter[] pakoja = new Leter[NumriILerave];
    private int letraAktuale = 0;
    // indeksi i Letres se ardhshme që do të (shpër)ndahet (0-51)

    // konstruktori e mbush pakon e Letrave
    public PakoELerave()
    {
        string[] vlerat = {"Pikë", "Dysh", "Tresh", "Katër", "Pesë", "Gjashtë",
                           "Shtatë", "Tetë", "Nëntë", "Dhjetë", "Xhandar", "Mbretëreshë", "Mbret"};
        string[] grupet = { "Zemër", "Diamant", "Rrush", "Gjeth" };

        // mbushë pakon me objekte Leter
        for (var numëruesi = 0; numëruesi < pakoja.Length; ++numëruesi)
        {
            pakoja[numëruesi] = new Leter(vlerat[numëruesi % 13], grupet[numëruesi / 13]);
        }
    }
}

```

```
// përzije pakon e Letrave me algoritëm me një kalim
// (one-pass algorithm)
public void Perzieji()
{
    // pas përzierjes, (shpër)ndarja duhet të nisë te pako[0] përsëri
    letraAktuale = 0; // ri-inicializojë letrënAktuale

    // për secilën Letër, zgjedhe një Letër tjetër të rastësishme
    // dhe shkëmbeji
    for (var ePara = 0; ePara < pakoja.Length; ++ePara)
    {
        // zgjedhe një numër të rastësishëm ndërmjet 0 dhe 51
        var eDyta = numratERastesishem.Next(NumriILetrave);

        // shkëmbeje Letrën aktuale me Letrën e zgjedhur rastësisht
        Leter temp = pakoja[ePara]; // temp = temporary, e përkohshme
        pakoja[ePara] = pakoja[eDyta];
        pakoja[eDyta] = temp;
    }
}

// (shpër)ndaje një Letër
public Leter NdajLeter()
{
    // përcaktoje se a kanë mbetur Letra për t'u ndarë
    if (letraAktuale < pakoja.Length)
    {
        return pakoja[letraAktuale++]; // ktheje Letrën aktuale në varg
    }
    else
    {
        return null; // trego se të gjitha Letrat janë ndarë
    }
}
}
```

```
// Fig. 8.12: TestIPakosSeLetrave.cs
// Aplikacion për përzierje dhe ndarje të letrave të lojës.
using System;

class TestIPakosSeLetrave
{
    // ekzekutoje aplikacionin
    static void Main()
    {
        var pakojaImeELeTrave = new PakoELeTrave();
        pakojaImeELeTrave.Perzieji(); // vendosi Letrat në radhë të rastësishme

        // shfaqti të 52 Letrat në radhën në të cilën janë (shpër)ndarë
        for (var i = 0; i < 52; ++i)
        {
            Console.WriteLine($"{pakojaImeELeTrave.NdajLeter()},{-24}");

            if ((i + 1) % 4 == 0)
            {
                Console.WriteLine();
            }
        }

        Console.ReadKey();
    }
}
```

Gjashtë - Diamant	Pesë - Rrush	Xhandar - Gjeth	Katër - Rrush
Mbretëreshë - Zemër	Tresh - Diamant	Nëntë - Rrush	Dysh - Gjeth
Pikë - Diamant	Tresh - Zemër	Pesë - Zemër	Shtatë - Diamant
Tresh - Gjeth	Tetë - Gjeth	Pikë - Zemër	Shtatë - Gjeth
Xhandar - Diamant	Mbret - Diamant	Mbretëreshë - Diamant	Tetë - Diamant
Katër - Gjeth	Mbret - Rrush	Dysh - Zemër	Pikë - Gjeth
Pikë - Rrush	Pesë - Gjeth	Nëntë - Diamant	Katër - Diamant
Mbretëreshë - Gjeth	Pesë - Diamant	Mbret - Zemër	Xhandar - Zemër
Mbretëreshë - Rrush	Tresh - Rrush	Nëntë - Zemër	Xhandar - Rrush
Dhjetë - Zemër	Gjashtë - Zemër	Shtatë - Zemër	Nëntë - Gjeth
Gjashtë - Gjeth	Dhjetë - Rrush	Mbret - Gjeth	Katër - Zemër
Dysh - Diamant	Tetë - Rrush	Shtatë - Rrush	Tetë - Zemër
Dhjetë - Diamant	Gjashtë - Rrush	Dhjetë - Gjeth	Dysh - Rrush

8.7 Pasimi (dërgimi, bartja) e vargjeve dhe elementeve të vargjeve te metodat

- Kur argumenti i një metode është varg apo element individual i një vargu të një tipi referencë, metoda e thirrur e pranon një kopje të referencës. Mirëpo, kur një argument për një metodë është element individual i vargut i tipit vlerë, metoda e thirrur e pranon një kopje të vlerës së elementit.

```
// Fig. 8.13: PasojeVargun.cs
// Pasimi i vargjeve dhe elementeve individuale të vargut - te metodat
using System;

class PasojeVargun
{
    // Main e krijon vargun dhe e thirr NdryshojeVargun dhe NdryshojeElementin
    static void Main()
    {
        int[] vargu = { 1, 2, 3, 4, 5 };

        Console.WriteLine("Efektet e pasimit të referencës së tërë vargut:");
        Console.WriteLine("Vlerat e vargut origjinal janë:");

        // afishoji elementet e vargut origjinal
        foreach (var vlera in vargu)
        {
            Console.Write($"    {vlera}");
        }

        NdryshojeVargun(vargu); // pasoje referencën e vargut
        Console.WriteLine("\n\nVlerat e vargut të ndryshuar janë:");

        // afishoji elementet e vargut të ndryshuar
        foreach (var vlera in vargu)
        {
            Console.Write($"    {vlera}");
        }

        Console.WriteLine("\n\nEfektet e pasimit të vlerës së elementit të vargut:\n" +
            $"vargu[3] para NdryshojeElementin: {vargu[3]}");

        NdryshojeElementin(vargu[3]); // provo ta ndryshosh elementin vargu[3]
        Console.WriteLine($"vargu[3] pas NdryshojeElementin: {vargu[3]}");

        Console.ReadKey();
    }

    // shumëzoje secilin element të një vargu me 2
    static void NdryshojeVargun(int[] vargu2)
    {
        for (var numeruesi = 0; numeruesi < vargu2.Length; ++numeruesi)
        {
            vargu2[numeruesi] *= 2;
        }
    }

    // shumëzoje argumentin me 2
    static void NdryshojeElementin(int elementi)
    {
        elementi *= 2;
        Console.WriteLine($"Vlera e elementit brenda NdryshojeElementin: {elementi}");
    }
}
```

```
Efektet e pasimit të referencës së tërë vargut:  
Vlerat e vargut origjinal janë:  
  1  2  3  4  5  
  
Vlerat e vargut të ndryshuar janë:  
  2  4  6  8  10  
  
Efektet e pasimit të vlerës së elementit të vargut:  
vargu[3] para NdryshojeElementin: 8  
Vlera e elementit brenda NdryshojeElementin: 16  
vargu[3] pas NdryshojeElementin: 8
```

8.8 Studim i rastit: ditari i notave që e përdor një varg për t'i ruajtur notat

```
Mirë se vini në ditarin e notave për lëndën  
SHK106 Programimi i Orientuar në Objekte!  
  
Notat janë:  
  
Studenti 1: 87  
Studenti 2: 68  
Studenti 3: 94  
Studenti 4: 100  
Studenti 5: 83  
Studenti 6: 78  
Studenti 7: 85  
Studenti 8: 91  
Studenti 9: 76  
Studenti 10: 87  
  
Mesatarja e klasës është 84.90  
Nota më e ulët është 68  
Nota më e lartë është 100  
  
Shpërndarja e notave:  
00-09:  
10-19:  
20-29:  
30-39:  
40-49:  
50-59:  
60-69: *  
70-79: **  
80-89: ****  
90-99: **  
100: *
```

```
// Fig. 8.15: DitarINotave.cs
// Ditar i notave që e përdor një varg për t'i ruajtur notat e testit.
using System;

class DitarINotave
{
    private int[] notat; // varg i notave të nxënësve/studentëve

    // properti EmriILendes që ka vetëm get (getter-only)
    // e implementuar automatikisht (auto-implemented)
    public string EmriILendes { get; }

    // konstruktori me dy parametra e inicializon
    // properti-n e implementuar automatikisht EmriILendes
    // dhe vargun e notave
    public DitarINotave(string emri, int[] varguINotave)
    {
        EmriILendes = emri; // caktoje proprietin EmriILendes të jetë emri
        notat = varguINotave; // inicializojë vargun e notave
    }

    // shfaq një mesazh të mirëpritjes për përdoruesin e DitaritTëNotave
    public void ShfaqMesazhin()
    {
        // properti EmriILendes e implementuar automatikisht e merr emrin e lëndës
        Console.WriteLine(
            $"Mirë se vini në ditarin e notave për lëndën\n{EmriILendes}!\n");
    }

    // kryej veprime të ndryshme në shënime
    public void ProcesojiNotat()
    {
        // shfaq në dalje (output) vargun e notave
        ShfaqNotat();

        // thirrë metodën MerreMesataren për ta llogaritur notën mesatare
        Console.WriteLine($"Mesatarja e klasës është {MerreMesataren():F}");

        // thirri metodat MerreMinimumin dhe MerreMaksimumin
        Console.WriteLine($"Nota më e ulët është {MerreMinimumin()}");
        Console.WriteLine($"Nota më e lartë është {MerreMaksimumin()}\n");

        // thirrë ShfaqBarChartin për ta shfaqur grafikun e shpërndarjes së notave
        ShfaqBarChartin();
    }

    // gjeje notën minimale
    public int MerreMinimumin()
    {
        var notaEUlet = notat[0]; // supozo se notat[0] është më e vogla

        // kalo nëpër vargun e notave
        foreach (var nota in notat)
        {
            // nëse nota është më e vogël se notaEUlet, caktoja atë variablës notaEUlet
            if (nota < notaEUlet)
            {
                notaEUlet = nota; // nota e re më e ulët
            }
        }
        return notaEUlet; // ktheje notën më të ulët
    }

    // gjeje notën maksimale
}
```

```
public int MerreMaksimumin()
{
    var notaELarte = notat[0]; // supozo se notat[0] është më e madhja

    // kalo nëpër vargun e notave
    foreach (var nota in notat)
    {
        // nëse nota është më e madhe se notaELarte, caktoja atë variablës notaELarte
        if (nota > notaELarte)
        {
            notaELarte = nota; // nota e re më e lartë
        }
    }

    return notaELarte; // ktheje notën më të lartë
}

// përcaktoje notën mesatare për testin
public double MerreMesataren()
{
    var totali = 0.0; // inicializojë totalin si double

    // mbledhi notat e studentëve
    foreach (var nota in notat)
    {
        totali += nota;
    }

    // ktheje mesataren e notave
    return totali / notat.Length;
}
```



```

// shfaq bar chart-in (grafikun me shirita) që e shfaq shpërndarjen e notave
public void ShfaqBarChartin()
{
    Console.WriteLine("Shpërndarja e notave:");

    // e ruan frekuencën e notave në secilin rang prej 10 notave
    var frekuenca = new int[11];

    // për secilën notë, rrite frekuencën përkatëse
    foreach (var nota in notat)
    {
        ++frekuenca[nota / 10];
    }

    // për secilën frekuencë të notave, shfaq bar-in (shiritin) në chart (grafik)
    for (var numeruesi = 0; numeruesi < frekuenca.Length; ++numeruesi)
    {
        // shfaq label-ën (etiketën) e bar-it (shiritit)
        // ("00-09: ", ..., "90-99: ", "100: ")
        if (numeruesi == 10)
        {
            Console.Write(" 100: ");
        }
        else
        {
            Console.Write($"{numeruesi * 10:D2}-{numeruesi * 10 + 9:D2}: ");
        }

        // shfaq shiritin (bar-in) e yjeve
        for (var yjet = 0; yjet < frekuenca[numeruesi]; ++yjet)
        {
            Console.Write("*");
        }

        Console.WriteLine(); // nise një rresht të ri të daljes (output-it)
    }
}

// shfaq përmbajtjen e vargut të notave
public void ShfaqNotat()
{
    Console.WriteLine("Notat janë:\n");

    // shfaq notën e secilit student
    for (var studenti = 0; studenti < notat.Length; ++studenti)
    {
        Console.WriteLine(
            $"Studenti {studenti + 1,2}: {notat[studenti],3}");
    }
}
}

```

8.9 Vargjet shumë-dimensionale

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0, 0]	a[0, 1]	a[0, 2]	a[0, 3]
Row 1	a[1, 0]	a[1, 1]	a[1, 2]	a[1, 3]
Row 2	a[2, 0]	a[2, 1]	a[2, 2]	a[2, 3]

Diagram illustrating a rectangular array with three rows and four columns. The array is labeled 'a'. The indices are shown as a[Row index, Column index]. The array name 'a' is indicated by an arrow pointing to the first element 'a[0, 0]'. The row index '0' is indicated by an arrow pointing to the first row. The column index '0' is indicated by an arrow pointing to the first column.

Fig. 8.17 | Rectangular array with three rows and four columns.

- Vargjet dydimensionale përdoren shpesh për t'i përfaqësuar tabelat e vlerave që përbëhen nga informacion i renditur në rreshta dhe shtylla. Për ta identifikuar një element të veçantë të tabelës, duhet t'i specifikojmë dy indeksa.
- C# i përkrah dy tipe të vargjeve dy-dimensionale – vargjet drejtkëndëshe (rectangular) dhe vargjet e dhëmbëzuara (jagged).

```
int[,] b = {{1, 2}, {3, 4}};
```

```
int[][] jagged = {new int[] {1, 2},
                  new int[] {3},
                  new int[] {4, 5, 6}};
```

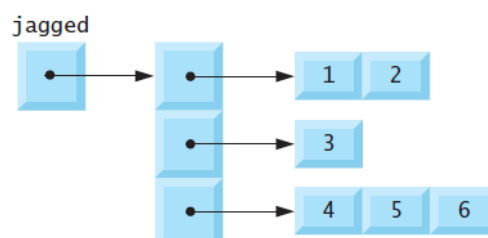


Fig. 8.18 | Jagged array with three rows of different lengths.

- Vargjet drejtkëndëshe përdoren për të përfaqësuar tabelat e informacionit në formë të rreshtave dhe shtyllave, ku secili rresht ka numër të njëjtë të shtyllave.
- Elementet në vargun drejtkëndësh **a** identifikohen me shprehje të formës **a[rreshti, shtylla]**.
- Një varg drejtkëndësh mund të krijohet me shprehjen për krijimin e vargut të formës

```
tipilVargut[,] emrilVargut = new tipilVargut[numrilRreshtave, numrilShtyllave];
```

- Një varg i dhëmbëzuar mirëmbahet si varg një-dimensional në të cilin secili element i referohet një vargu një-dimensional.
- Gjatësitë e rreshtave në një varg të dhëmbëzuar nuk duhet të jenë të njëjta.
- Elementeve në vargun e dhëmbëzuar *emrilVargut* mund t'u qasemi me një shprehje për qasje në varg të formës

```
emrilVargut[rreshti][shtylla]
```

- Vargu i dhëmbëzuar mund të deklarohet dhe inicializohet në formën:

```
tipilVargut[ ] [ ] emrilVargut = {new tipilVargut[ ] {inicializuesi i rreshtit0},  
                                {new tipilVargut[ ] {inicializuesi i rreshtit1}, ...};
```

```
// Fig. 8.19: InicVargjet.cs
// Inicializimi i vargjeve drejtkëndëshe dhe të dhëmbëzuara.
using System;

class InicVargjet
{
    // krijo dhe shfaq vargje drejtkëndësh dhe të dhëmbëzuar
    static void Main()
    {
        // me vargje drejtkëndësh,
        // secili rresht duhet të ketë gjatësi të njëjtë
        int[,] drejtkendeshi = { { 1, 2, 3 }, { 4, 5, 6 } };

        // me vargje të dhëmbëzuara
        // duhet ta përdorim "new int[]" për secili rresht,
        // por nuk duhet që secili rresht të jetë me gjatësi të njëjtë
        int[][] idhembezuari = {new int[] {1, 2},
                                new int[] {3},
                                new int[] {4, 5, 6}};

        ShfaqVargun(drejtkendeshi); // e shfaq vargun drejtkëndësh sipas rreshtit
        Console.WriteLine(); // shfaq një rresht të zbrazët
        ShfaqVargun(idhembezuari); // e shfaq vargun e dhëmbëzuar sipas rreshtit

        Console.ReadKey();
    }

    // shfaq rreshtat dhe shtyllat (kolonat)
    // e një vargu drejtkëndësh
    static void ShfaqVargun(int[,] vargu)
    {
        Console.WriteLine("Vlerat në vargun drejtkëndësh sipas rreshtit janë");

        // kalo nëpër rreshtat e vargut
        for (var rreshti = 0; rreshti < vargu.GetLength(0); ++rreshti)
        {
            // kalo nëpër shtyllat e rreshtit aktual
            for (var shtylla = 0; shtylla < vargu.GetLength(1); ++shtylla)
            {
                Console.Write($"{vargu[rreshti, shtylla]} ");
            }

            Console.WriteLine(); // nise një rresht të ri të daljes (output-it)
        }
    }

    // shfaq rreshtat dhe shtyllat e një vargu të dhëmbëzuar
    static void ShfaqVargun(int[][] vargu)
    {
        Console.WriteLine("Vlerat në vargun e dhëmbëzuar sipas rreshtit janë");

        // kalo nëpër secilin rresht
        foreach (var rreshti in vargu)
        {
            // kalo nëpër secilin element në rreshtin aktual
            foreach (var elementi in rreshti)
            {
                Console.Write($"{elementi} ");
            }

            Console.WriteLine(); // nise një rresht të ri të daljes (output-it)
        }
    }
}
```

```
Vlerat në vargun drejtkëndësh sipas rreshtit janë
1 2 3
4 5 6

Vlerat në vargun e dhëmbëzuar sipas rreshtit janë
1 2
3
4 5 6
```

8.10 Studim i rastit: DitarINotave duke përdorur varg drejtkëndësh

- Kur urdhri foreach i përshkon (kalon nëpër) elementet e vargut drejtkëndësh, e shikon çdo element të rreshtit të parë me radhë sipas indeksit, pastaj secilin element të rreshtit të dytë me radhë sipas indeksit e kështu me radhë.

```
Mirë se vini në ditarin e notave për lëndën
SHK106 Programimi i Orientuar në Objekte!

Notat janë:

      Testi 1  Testi 2  Testi 3  Mesatarja
Studenti 1      87      96      70      84.33
Studenti 2      68      87      90      81.67
Studenti 3      94     100      90      94.67
Studenti 4     100      81      82      87.67
Studenti 5      83      65      85      77.67
Studenti 6      78      87      65      76.67
Studenti 7      85      75      83      81.00
Studenti 8      91      94     100      95.00
Studenti 9      76      72      84      77.33
Studenti 10     87      93      73      84.33

Nota më e ulët në ditar është 65
Nota më e lartë në ditar është 100

Shpërndarja e përgjithshme e notave:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: ***
70-79: *****
80-89: *****
90-99: *****
100: ***
```

```
// Fig. 8.21: DitarINotave.cs
// Ditar i notave që e përdor një varg drejtkëndësh për t'i ruajtur notat
using System;

class DitarINotave
{
    private int[,] notat; // varg drejtkëndësh i notave të studentëve

    // properti e implementuar automatikisht (auto-implemented) EmriILendes
    public string EmriILendes { get; }

    // konstruktori me dy parametra e inicializon
    // proprietin e implementuar automatikisht EmriILendes dhe vargun e notave
    public DitarINotave(string emri, int[,] varguINotave)
    {
        EmriILendes = emri; // caktoje EmriILendes të jetë emri
        notat = varguINotave; // inicializojë vargun e notave
    }

    // shfaqja një mesazh mirëpritës përdoruesit të DitaritTëNotave
    public void ShfaqeMesazhin()
    {
        // properti e implementuar automatikisht EmriILendes e merr emrin e lëndës
        Console.WriteLine(
            $"Mirë se vini në ditarin e notave për lëndën\n{EmriILendes}!\n");
    }

    // kryej veprime (operacione) të ndryshme në shënime
    public void ProcesojiNotat()
    {
        // shfaq vargun e notave
        ShfaqNotat();

        // thirri metodat MerreMinimumin dhe MerreMaksimumin
        Console.WriteLine(
            $"\\nNota më e ulët në ditar është {MerreMinimumin()}\" +
            $"\\nNota më e lartë në ditar është {MerreMaksimumin()}\\n");

        // shfaq grafikun (chart-in) e shpërndarjes së të gjitha notave në të gjitha testet
        ShfaqBarChartin();
    }

    // gjeje notën minimale
    public int MerreMinimumin()
    {
        // supozo se elementi i parë i vargut notat është më i vogli
        var notaEUlet = notat[0, 0];

        // kalo (loop) nëpër elementet e vargut drejtkëndësh të notave
        foreach (var nota in notat)
        {
            // nëse nota është më e ulët se notaEUlet, caktoja atë ndryshores notaEUlet
            if (nota < notaEUlet)
            {
                notaEUlet = nota;
            }
        }

        return notaEUlet; // ktheje notën më të ulët
    }
}
```

```
// gjeje notën maksimale
public int MerreMaksimumin()
{
    // supozo se elementi i parë i vargut notat është më i madhi
    var notaELarte = notat[0, 0];

    // kalo (loop) nëpër elementet e vargut drejtkëndësh të notave
    foreach (var nota in notat)
    {
        // nëse nota është më e madhe se notaELarte, caktoja atë ndryshores notaELarte
        if (nota > notaELarte)
        {
            notaELarte = nota;
        }
    }

    return notaELarte; // ktheje notën më të lartë
}

// përcaktoje notën mesatare për një student të caktuar
public double MerreMesataren(int studenti)
{
    // merre numrin e notave për student
    var numriINotave = notat.GetLength(1);
    var totali = 0.0; // inicializojë totalin

    // mbledhi notat për studentin
    for (var provimi = 0; provimi < numriINotave; ++provimi)
    {
        totali += notat[studenti, provimi];
    }

    // ktheje mesataren e notave
    return totali / numriINotave;
}
```

```
// shfaq bar chart-in (grafikun me shirita) që e paraqet shpërndarjen e përgjithshme të notave
public void ShfaqBarChartin()
{
    Console.WriteLine("Shpërndarja e përgjithshme e notave:");

    // e ruan frekuencën e notave në secilin rang prej 10 notash
    var frekuenca = new int[11];

    // për secilën notë në objektin DitarINotave, rrite frekuencën përkatëse
    foreach (var nota in notat)
    {
        ++frekuenca[nota / 10];
    }

    // për secilën frekuencë të notave, shfaq bar-in (shiritin) në chart (grafik)
    for (var numri = 0; numri < frekuenca.Length; ++numri)
    {
        // shfaq etiketën e bar-it ("00-09: ", ..., "90-99: ", "100: ")
        if (numri == 10)
        {
            Console.Write(" 100: ");
        }
        else
        {
            Console.Write($"{numri * 10:D2}-{numri * 10 + 9:D2}: ");
        }

        // shfaq bar-in (shiritin) e yjeve
        for (var yjet = 0; yjet < frekuenca[numri]; ++yjet)
        {
            Console.Write("*");
        }

        Console.WriteLine(); // filloje një rresht të ri të shfaqjes (daljes, output)
    }
}
```



```
// shfaqe përmbajtjen e vargut të notave
public void ShfaqNotat()
{
    Console.WriteLine("Notat janë:\n");
    Console.Write(" "); // rreshtoji ballinat e shtyllave

    // krijoje një ballinë të shtyllës për secilin test
    for (var testi = 0; testi < notat.GetLength(1); ++testi)
    {
        Console.Write($"Testi {testi + 1} ");
    }

    Console.WriteLine("Mesatarja"); // ballina e shtyllës për mesataren e studentit

    // krijoji rreshtat/shtyllat e tekstit që e paraqet vargun notat
    for (var studenti = 0; studenti < notat.GetLength(0); ++studenti)
    {
        Console.Write($"Studenti {studenti + 1,2}");

        // shfaq notat e studentit
        for (var nota = 0; nota < notat.GetLength(1); ++nota)
        {
            Console.Write($" {notat[studenti, nota],9}");
        }

        // thirre metodën MerreMesataren
        // për ta llogaritur notën mesatare të studentit;
        // pasoje (kaloja) numrin e rreshtit si argument metodës MerreMesataren
        Console.WriteLine($" {MerreMesataren(studenti),9:F}");
    }
}
```

```
// Fig. 8.22: TestiIDitaritTeNotave.cs
// Krijoje një objekt DitarINotave duke e përdorur një varg drejtkëndësh të
// notave.
using System;

class TestiIDitaritTeNotave
{
    // Metoda Main e nis ekzekutimin e aplikacionit
    static void Main()
    {
        // varg drejtkëndësh i notave të 10 studentëve
        int[,] varguINotave = {{87, 96, 70},
                                {68, 87, 90},
                                {94, 100, 90},
                                {100, 81, 82},
                                {83, 65, 85},
                                {78, 87, 65},
                                {85, 75, 83},
                                {91, 94, 100},
                                {76, 72, 84},
                                {87, 93, 73}};

        DitarINotave ditariImINotave = new DitarINotave(
            "SHK106 Programimi i Orientuar në Objekte", varguINotave);
        ditariImINotave.ShfaqeMesazhin();
        ditariImINotave.ProcesojiNotat();
        Console.ReadKey();
    }
}
```

Notat janë:

	Testi 1	Testi 2	Testi 3	Mesatarja
Studenti 1	87	96	70	84.33
Studenti 2	68	87	90	81.67
Studenti 3	94	100	90	94.67
Studenti 4	100	81	82	87.67
Studenti 5	83	65	85	77.67
Studenti 6	78	87	65	76.67
Studenti 7	85	75	83	81.00
Studenti 8	91	94	100	95.00
Studenti 9	76	72	84	77.33
Studenti 10	87	93	73	84.33

Nota më e ulët në ditar është 65
 Nota më e lartë në ditar është 100

Shpërndarja e përgjithshme e notave:

```
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: ***
70-79: *****
80-89: *****
90-99: *****
100: ***
```

8.11 Listat e argumenteve me gjatësi të ndryshueshme (variable, variable-length)

- Një parametër varg një-dimensional i paraprirë nga **params** në listën e parametrave të metodës tregon se metoda e pranon një numër të ndryshueshëm të argumenteve me tipin e elementeve të vargut.
- Ndryshuesi (modifikuesi, modifier) **params** mund të shfaqet vetëm në elementin e fundit të listës së parametrave.
- C# e trajton një listë të argumenteve me gjatësi të ndryshueshme si varg një-dimensional.

```
// Fig. 8.23: TestIVargutParameter.cs
// Përdorimi i listave të argumenteve me gjatësi të ndryshueshme.
using System;

class TestIVargutParameter
{
    // llogarite mesataren
    static double Mesatarja(params double[] numrat)
    {
        var totali = 0.0; // inicializojë totalin

        // llogarite totalin duke e përdorur urdhrin foreach
        foreach (var d in numrat)
        {
            totali += d;
        }

        return numrat.Length != 0 ? totali / numrat.Length : 0.0;
    }

    static void Main()
    {
        var d1 = 10.0;
        var d2 = 20.0;
        var d3 = 30.0;
        var d4 = 40.0;

        Console.WriteLine(
            $"d1 = {d1:F1}\nd2 = {d2:F1}\nd3 = {d3:F1}\nd4 = {d4:F1}\n");
        Console.WriteLine($"Mesatarja e d1 dhe d2 është {Mesatarja(d1, d2):F1}");
        Console.WriteLine(
            $"Mesatarja e d1, d2 dhe d3 është {Mesatarja(d1, d2, d3):F1}");
        Console.WriteLine(
            $"Mesatarja e d1, d2, d3 dhe d4 është {Mesatarja(d1, d2, d3, d4):F1}");
        Console.ReadKey();
    }
}
```

```
d1 = 10.0
d2 = 20.0
d3 = 30.0
d4 = 40.0

Mesatarja e d1 dhe d2 është 15.0
Mesatarja e d1, d2 dhe d3 është 20.0
Mesatarja e d1, d2, d3 dhe d4 është 25.0
```

8.12 Përdorimi i argumenteve command-line (në dritaren komanduese)

- Kur një aplikacion ekzekutohet nga Command Prompt (Dritarja komanduese), ambienti i ekzekutimit ia pason argumentet që shfaqen pas emrit të aplikacionit – metodës Main të aplikacionit – si **string**je në një varg një-dimensional.
- Në Visual Studio, argumentet command-line jepen te Projekti -> Properties -> Debug -> Command line arguments.

```
// Fig. 8.24: InicVargun.cs
// Përdorimi i argumenteve command-line (në dritaren komanduese)
// për ta inicializuar një varg.
using System;

class InicVargun
{
    static void Main(string[] args)
    {
        // kontrolloje numrin e argumenteve command-line
        if (args.Length != 3)
        {
            Console.WriteLine(
                "Gabim: Ju lutem ri-shkruajeni tërë komandën, duke e përfshirë\n" +
                "madhësinë e vargut, vlerën fillestare dhe shtesën (inkrementin).");
        }
        else
        {
            // merre madhësinë e vargut nga argumenti i parë command-line
            var gjatesiaEVargut = int.Parse(args[0]);
            var vargu = new int[gjatesiaEVargut]; // krijoje vargun

            // merre vlerën fillestare dhe shtesën (inkrementin) nga argumenti command-line
            var vleraFillestare = int.Parse(args[1]);
            var inkrementi = int.Parse(args[2]);

            // llogarite vlerën për secilin element të vargut
            for (var numeruesi = 0; numeruesi < vargu.Length; ++numeruesi)
            {
                vargu[numeruesi] = vleraFillestare + inkrementi * numeruesi;
            }

            Console.WriteLine($"{ "Indeksi" } { "Vlera", 8 }");

            // shfaqe indeksin dhe vlerën e vargut
            for (int numeruesi = 0; numeruesi < vargu.Length; ++numeruesi)
            {
                Console.WriteLine($"{ numeruesi, 5 } { vargu[numeruesi], 8 }");
            }
        }
        Console.ReadKey();
    }
}
```

Indeksi	Vlera
0	100
1	110
2	120
3	130
4	140

8.13 (Opsionale) Pasimi i vargjeve me vlerë dhe me referencë

- Kur një objekt i tipit referencë pasohet me **ref**, metoda e thirrur në fakt fiton kontroll ndaj vetë referencës, duke ia lejuar metodës së thirrur ta zëvendësojë referencën origjinale në thirrësin me një objekt tjetër apo madje edhe me **null**.
- Nëse e hasni një situatë ku vërtet dëshironi që procedura e thirrur ta ndryshojë referencën e thirrësit, pasojeni parametrin e tipit referencë duke e përdorur fjalën kyçe **ref** – por situatat e tilla janë të rralla.

```
// Fig. 8.25: TestIReferencesSeVargut.cs
// Testimi i efekteve të pasimit (kalimit)
// të referencave të vargut me vlerë dhe me referencë
using System;

class TestIReferencesSeVargut
{
    static void Main()
    {
        // krijoje dhe inicializojë vargun e pare
        int[] varguIPare = { 1, 2, 3 };

        // kopjoje referencën në variablën varguIPare
        int[] kopjaEVargutTePare = varguIPare;

        Console.WriteLine("Testoje pasimin e referencës së vargut të parë me vlerë");
        Console.Write(
            "Përmbajtja e vargut të parë para thirrjes së metodës DyfishimiIPare:\n\t");

        // shfaqë përmbajtjen e variablës varguIPare
        foreach (var elementi in varguIPare)
        {
            Console.Write($"{elementi} ");
        }

        // pasoja (kaloja) variablën varguIPare me vlerë - metodës DyfishimiIPare
        DyfishimiIPare(varguIPare);

        Console.Write(
            "\nPërmbajtja e variablës varguIPare pas thirrjes së metodës DyfishimiIPare:\n\t");

        // shfaqë përmbajtjen e variablës varguIPare
        foreach (var elementi in varguIPare)
        {
            Console.Write($"{elementi} ");
        }
    }
}
```

```
// testoje se a është ndryshuar referenca nga DyfishimiIPare
if (varguIPare == kopjaEVargutTePare)
{
    Console.WriteLine("\n\nReferencat i referohen vargut të njëjtë");
}
else
{
    Console.WriteLine(
        "\n\nReferencat u referohen vargjeve të ndryshme");
}

// krijoje dhe inicializojë vargun e dytë
int[] varguIDyte = { 1, 2, 3 };

// kopjoje referencën në variablën varguIDyte
int[] kopjaEVargutTeDyte = varguIDyte;

Console.WriteLine(
    "\nTestoje pasimin (kalimin) e referencës së vargut të dytë me referencë");
Console.Write(
    "Përmbajtja e variablës varguIDyte para thirrjes së metodës DyfishimiIDyte:\n\t");

// shfaqë përmbajtjen e variablës varguIDyte para thirrjes së metodës
foreach (var elementi in varguIDyte)
{
    Console.Write($"{elementi} ");
}

// pasoja (kaloja) variablën varguIDyte me referencë metodës DyfishimiIDyte
DyfishimiIDyte(ref varguIDyte);

Console.Write(
    "\nPërmbajtja e variablës varguIDyte pas thirrjes së metodës DyfishimiIDyte:\n\t");

// shfaqë përmbajtjen e variablës varguIDyte pas thirrjes së metodës
foreach (var elementi in varguIDyte)
{
    Console.Write($"{elementi} ");
}

// testoje se a është ndryshuar referenca nga DyfishimiIDyte
if (varguIDyte == kopjaEVargutTeDyte)
{
    Console.WriteLine("\n\nReferencat i referohen vargut të njëjtë");
}
else
{
    Console.WriteLine(
        "\n\nReferencat u referohen vargjeve të ndryshme");
}

Console.ReadKey();
}
```

```

// ndryshoji elementet e vargut dhe provo ta ndryshosh referencën
static void DyfishimiIPare(int[] vargu)
{
    // dyfishoje vlerën e secilit element
    for (var i = 0; i < vargu.Length; ++i)
    {
        vargu[i] *= 2;
    }

    // krijoje një objekt të ri dhe caktoja referencën e tij ndryshores vargu
    vargu = new int[] { 11, 12, 13 };
}

// ndryshoji elementet e vargut dhe ndryshoje referencën e vargut
// që t'i referohet një vargu të ri
static void DyfishimiIDyte(ref int[] vargu)
{
    // dyfishoje vlerën e secilit element
    for (var i = 0; i < vargu.Length; ++i)
    {
        vargu[i] *= 2;
    }

    // krijoje një objekt të ri dhe caktoja referencën e tij ndryshores vargu
    vargu = new int[] { 11, 12, 13 };
}
}

```

```

Testoje pasimin e referencës së vargut të parë me vlerë
Përmbajtja e vargut të parë para thirrjes së metodës DyfishimiIPare:
    1 2 3
Përmbajtja e variablës varguIPare pas thirrjes së metodës DyfishimiIPare:
    2 4 6

Referencat i referohen vargut të njëjtë

Testoje pasimin (kalimin) e referencës së vargut të dytë me referencë
Përmbajtja e variablës varguIDyte para thirrjes së metodës DyfishimiIDyte:
    1 2 3
Përmbajtja e variablës varguIDyte pas thirrjes së metodës DyfishimiIDyte:
    11 12 13

Referencat u referohen vargjeve të ndryshme

```


9. Hyrje në LINQ dhe në koleksionin List

9.1 Hyrje

- Klasat e koleksioneve të .NET-it ofrojnë struktura të të ripërdorshme të shënimeve që janë të besueshme, të fuqishme dhe efikase.
- **Listat** e rrisin automatikisht madhësinë e tyre për të akomoduar elemente shtesë.
- Sasi të mëdha të shënimeve shpesh ruhen në një bazë të shënimeve – një koleksion i organizuar i shënimeve. Sistemet e sotme më të njohura të bazës së shënimeve janë bazat relacionale të shënimeve. SQL (Structured Query Language – Gjuha e Strukturuar për Kërkesa/Pyetje) është gjuha standarde ndërkombëtare që përdoret pothuajse universalisht me bazat relacionale të shënimeve për të bërë kërkesa/pyetje (domethënë për të kërkuar informacion që i plotëson kriteret e dhëna.)
- LINQ ju lejon të shkruani shprehje të query-ve / kërkesave (të ngjashme me query-t SQL) që marrin informacion nga një shumëllojshmëri e gjerë e burimeve të shënimeve. Mund t’u bëni query vargjeve dhe **Listave**, duke i zgjedhur elementet që e plotësojnë një set (grup) të kushteve – kjo njihet si filtrim.
- Një provajder (provider, ofrues, furnizues) i LINQ është një set (grup) i klasave që i implementojnë operacionet LINQ dhe ua mundësojmë programeve të ndërveprojnë me burimet e shënimeve për të kryer punë të tilla si sortimi (renditja), grupimi dhe filtrimi i elementeve.

9.2 Bërja e query-ve në një varg të vlerave int duke përdorur LINQ

- Urdhrrat e iterimit fokusohen në procesin e iterimit nëpër elemente. LINQ i specifikon kushtet që duhet t’i plotësojnë elementet e zgjedhura, e jo hapat e nevojshëm për t’i marrë rezultatet.
- Namespace-i (hapësira emërore) **System.Linq** i përmban klasat për LINQ to Objects (LINQ në Objekte).

```
// Fig. 9.2: LINQMeVargMeTipTeThjeshte.cs
// LINQ to Objects (LINQ në Objekte) duke e përdorur një varg të int-ave.
using System;
using System.Linq;

class LINQMeVargMeTipTeThjeshte
{
    static void Main()
    {
        // krijoje një varg të numrave të plotë
        var vlerat = new[] { 2, 9, 5, 0, 3, 7, 1, 4, 8, 5 };

        // shfaq vlerat origjinale
        Console.WriteLine("Vargu origjinal (fillestar):");
        foreach (var elementi in vlerat)
        {
            Console.WriteLine($"{elementi}");
        }

        // query në LINQ që i nxjerr vlerat më të mëdha se 4 nga vargu
        var vleratEFiltruara =
            from vlera in vlerat // burimi i shënimeve është vargu vlerat
            where vlera > 4
            select vlera;

        // shfaq rezultatet për vleratEFiltruara
        Console.WriteLine("\nVargu me vlerat më të mëdha se 4:");
        foreach (var elementi in vleratEFiltruara)
        {
            Console.WriteLine($"{elementi}");
        }

        // përdore klauzolën orderby për t'i sortuar (radhitur)
        // vlerat fillestare (origjinale) në renditje rritëse
        var vleratESortuara =
            from vlera in vlerat // burimi i shënimeve është vargu vlerat
            orderby vlera
            select vlera;

        // shfaq rezultatet për vleratESortuara
        Console.WriteLine("\nVargu origjinal, vleratESortuara:");
        foreach (var elementi in vleratESortuara)
        {
            Console.WriteLine($"{elementi}");
        }

        // sortoji (renditi) rezultatet për vleratEFiltruara në renditje rënëse
        var rezultatetEFiltruaraTeSortuara =
            from vlera in vleratEFiltruara
            // burimi i shënimeve është query në LINQ vleratEFiltruara
            orderby vlera descending
            select vlera;

        // shfaq rezultatet për rezultatetEFiltruaraTeSortuara
        Console.WriteLine(
            "\nVlerat më të mëdha se 4, renditja rënëse (dy query):");
        foreach (var elementi in rezultatetEFiltruaraTeSortuara)
        {
            Console.WriteLine($"{elementi}");
        }
    }
}
```

```
// filtroje vargun origjinal dhe sortoji rezultatet në radhë rënëse
var vleratESortuaraDheTeFiltruara =
    from vlera in vlerat // burimi i shënimeve është vargu vlerat
    where vlera > 4
    orderby vlera descending
    select vlera;

// shfaq rezultate për vleratEFiltruara dhe për vleratESortuara
Console.Write(
    "\nVlerat më të mëdha se 4, renditja rënëse (një query):");
foreach (var elementi in vleratESortuaraDheTeFiltruara)
{
    Console.Write($" {elementi}");
}

Console.WriteLine();
Console.ReadKey();
}
```

```
Vargu origjinal (fillestar): 2 9 5 0 3 7 1 4 8 5
Vargu me vlerat më të mëdha se 4: 9 5 7 8 5
Vargu origjinal, vleratESortuara: 0 1 2 3 4 5 5 7 8 9
Vlerat më të mëdha se 4, renditja rënëse (dy query): 9 8 7 5 5
Vlerat më të mëdha se 4, renditja rënëse (një query): 9 8 7 5 5
```

9.2.1 Klauzola from

- Klauzola **from** e specifikon një variabël të rangut dhe burimin e shënimeve për t'i bërë query. Variabla e rangut e përfaqëson secilin element në burimin e shënimeve (një nga një).

9.2.2 Klauzola where

- Nëse kushti në klauzolën **where** vlerësohet si **true** për një element, ai përfshihet në rezultate.

9.2.3 Klauzola select

- Klauzola **select** e specifikon një shprehje që e përcakton se çfarë vlere shfaqet në rezultate.

9.2.4 Iterimi nëpër rezultatet e query-t LINQ

- Një urdhër **foreach** mund të iterojë nëpër çfarëdo objekti që e implementon **IEnumerable<T>**.

9.2.5 Klauzola orderby

- Klauzola **orderby** i sorton (rendit) rezultatet e query-t, duke u bazuar në një apo më shumë shprehje, zakonisht në renditje rritëse (by default). Rezultatet mund të sortohen edhe në renditje zbritëse duke e përdorur modifikuesin **descending**.

9.2.6 Interfejsi (ndërfaqja) IEnumerable<T>

- Shumica e query-ve LINQ e kthejnë një objekt që e implementon interfejsin **IEnumerable<T>**.
- Një interfejs në C# i përshkruan anëtarët që mund të përdoren për të ndërvepruar me një objekt.
- Interfejsi **IEnumerable<T>** e përshkruan funksionalitetin e çdo objekti që e mundëson të iterohet nëpër të dhe kështu ofron metoda për t'iu qasur secilit element në sekuencë.
- Një klasë që e implementon një interfejs duhet ta definojë secilin element në interface.
- Vargjet dhe koleksionet gjenerike e implementojnë interfejsin **IEnumerable<T>**.

9.3 Bërja e query-ve në një varg të objekteve Punonjës duke përdorur LINQ

- LINQ mund të përdoret me koleksione të çfarëdo tipi të shënimeve.

```
// Fig. 9.3: Punonjes.cs
// Klasa Punonjes me proprietit Emri, Mbiemri dhe PagaMujore.
class Punonjes
{
    public string Emri { get; }
    // properti vetëm për lexim e implementuar automatikisht
    public string Mbiemri { get; }
    // properti vetëm për lexim e implementuar automatikisht
    private decimal pagaMujore; // paga mujore e punonjësit

    // konstruktori e inicializon emrin, mbiemrin dhe pagën mujore
    public Punonjes(string emri, string mbiemri,
        decimal pagaMujore)
    {
        Emri = emri;
        Mbiemri = mbiemri;
        PagaMujore = pagaMujore;
    }

    // properti që e merr (gets) dhe e cakton (sets)
    // pagën mujore të punonjësit
    public decimal PagaMujore
    {
        get
        {
            return pagaMujore;
        }
        set
        {
            if (value >= 0M) // validoje që paga është jonegative
            {
                pagaMujore = value;
            }
        }
    }

    // ktheje një string që e përmban informacionin e punonjësit
    public override string ToString() =>
        $"{Emri,-10} {Mbiemri,-10} {PagaMujore,10:C}";
}
```

```

// Fig. 9.4: LINQmeVargTeObjekteve.cs
// LINQ to Objects (LINQ në Objekte)
// duke përdorur një varg të objekteve Punonjes.
using System;
using System.Linq;

class LINQmeVargTeObjekteve
{
    static void Main()
    {
        // inicializojë vargun p punonjësve
        var punonjesit = new[] {
            new Punonjes("Jason", "Red", 5000M),
            new Punonjes("Ashley", "Green", 7600M),
            new Punonjes("Matthew", "Indigo", 3587.5M),
            new Punonjes("James", "Indigo", 4700.77M),
            new Punonjes("Luke", "Indigo", 6200M),
            new Punonjes("Jason", "Blue", 3200M),
            new Punonjes("Wendy", "Brown", 4236.4M)};

        // shfaqni të gjithë punonjësit
        Console.WriteLine("Vargu origjinal (fillestar):");
        foreach (var elementi in punonjesit)
        {
            Console.WriteLine(elementi);
        }

        // filtroje një rang të pagave duke përdorur && në një LINQ-query
        var ndermjet4Ke6K =
            from p in punonjesit
            where (p.PagaMujore >= 4000M) && (p.PagaMujore <= 6000M)
            select p;

        // shfaqni punonjësit që fitojnë ndërmjet 4000 p 6000 në muaj
        Console.WriteLine("\nPunonjësit që fitojnë në rangun " +
            $"{4000:C}-{6000:C} në muaj:");
        foreach (var elementi in ndermjet4Ke6K)
        {
            Console.WriteLine(elementi);
        }

        // radhiti punonjësit sipas mbiemrit, pastaj emrit me LINQ
        var sortuarMeEmer =
            from p in punonjesit
            orderby p.Mbiemri, p.Emri
            select p;

        // ballina (header-i)
        Console.WriteLine("\nPunonjësi i parë kur sortohet sipas emrit:");

        // provo ta shfaqësh rezultatin p parë të LINQ-query-t të mësipërm
        // attempt to display the first result of the above LINQ query
        if (sortuarMeEmer.Any())
        {
            Console.WriteLine(sortuarMeEmer.First());
        }
        else
        {
            Console.WriteLine("nuk u gjet");
        }
    }
}

```

```

// përdore LINQ për t'i zgjedhur (select)
// mbiemrat p punonjësve
var mbiemrat =
    from p in punonjesit
    select p.Mbiemri;

// përdore metodën Distinct për t'i zgjedhur
// mbiemrat unikë
Console.WriteLine("\nMbiemrat unikë të punonjësve:");
foreach (var elementi in mbiemrat.Distinct())
{
    Console.WriteLine(elementi);
}

// përdore LINQ-un për t'i zgjedhur emrat dhe mbiemrat
var emrat =
    from p in punonjesit
    select new { p.Emri, p.Mbiemri };

// shfaq emrat e plotë
Console.WriteLine("\nVetëm emrat:");
foreach (var elementi in emrat)
{
    Console.WriteLine(elementi);
}

Console.WriteLine();
Console.ReadKey();
}
}

```

```

Vargu origjinal (fillestari):
Jason      Red      £5,000.00
Ashley     Green     £7,600.00
Matthew    Indigo    £3,587.50
James      Indigo    £4,700.77
Luke       Indigo    £6,200.00
Jason      Blue      £3,200.00
Wendy      Brown     £4,236.40

Punonjësit që fitojnë në rangun £4,000.00-£6,000.00 në muaj:
Jason      Red      £5,000.00
James      Indigo    £4,700.77
Wendy      Brown     £4,236.40

Punonjësi i parë kur sortohet sipas emrit:
Jason      Blue      £3,200.00

Mbiemrat unikë të punonjësve:
Red
Green
Indigo
Blue
Brown

Vetëm emrat:
{ Emri = Jason, Mbiemri = Red }
{ Emri = Ashley, Mbiemri = Green }
{ Emri = Matthew, Mbiemri = Indigo }
{ Emri = James, Mbiemri = Indigo }
{ Emri = Luke, Mbiemri = Indigo }
{ Emri = Jason, Mbiemri = Blue }
{ Emri = Wendy, Mbiemri = Brown }

```

9.3.2 Sortimi i rezultateve të një query LINQ sipas shumë properti-ve

- Një klauzolë **orderby** mund t'i sortojë rezultatet sipas shprehjeve të shumta të specifikuara në një listë të ndarë me presje.

9.3.3 Metodatat e zgjerimit Any, First dhe Count

- Metoda **Any** kthen **true** nëse është të paktën një element në query të cilit i aplikohet; përndryshe, kthen **false**.
- Metoda **First** e kthen elementin e parë të query-t të cilës i aplikohet. Duhet ta kontrolloni që rezultati i query-t nuk është i zbrazët para se ta thirrni **First**.
- Metoda **Count** e kthen numrin e elementeve në query-n të cilës i aplikohet.

9.3.4 Zgjedhja e një properti-e të një objekti

- Metoda **Distinct** i heq vlerat e dyfishta nga rezultatet e query-t (pyetësorit).

9.3.5 Krijimi i tipeve të reja në klauzolën select të një query LINQ

- Në një klauzolë **select**, mund ta zgjedhësh çdo numër të properti-ve (një projeksion) dhe t'i enkapsulosh ato në objekte duke i specifikuar ato në një listë të ndarë me presje në kllapa gjarpërore pas fjalës kyçe **new**. Kompajleri automatikisht e krijon një klasë të re që i ka këto properti – të quajtur tip anonim.

9.4 Hyrje në koleksione

- Klasat e koleksioneve në .NET ofrojnë metoda efikase që i organizojnë, i ruajnë dhe i marrin shënimet pa kërkuar njohuri se si po ruhen shënimet.

9.4.1 Koleksioni List<T>

- Klasa **List<T>** është e ngjashme me një varg por ofron funksionalitet më të pasur, si përshembull ndryshimi dinamik i madhësisë (dynamic resizing).
- Metoda **Add** e shton një element në fund të një **Liste**.
- Metoda **Insert** e fut një element në një pozicion të caktuar në **Listë**.
- Property **Count** e kthen numrin e elementeve që janë aktualisht në një **Listë**.
- **Listat** mund të indeksohen si vargjet duke e vënë indeksin në kllapa të mesme (katrore) pas emrit të objektit.
- Metoda **Remove** përdoret për ta hequr elementin e parë me një vlerë specifike.
- Metoda **RemoveAt** e heq elementin në indeksin e specifikuar.
- Metoda **Contains** kthen **true** nëse elementi gjendet në **Listë**, dhe **false** përndryshe.
- Properti **Capacity** tregon se sa elemente mund t'i mbajë një **Listë** pa u rritur.

Method or property	Description
Add	Adds an element to the end of the List.
AddRange	Adds the elements of its collection argument to the end of the List.
Capacity	Property that <i>gets</i> or <i>sets</i> the number of elements a List can store without resizing.
Clear	Removes all the elements from the List.
Contains	Returns <code>true</code> if the List contains the specified element and <code>false</code> otherwise.
Count	Property that returns the number of elements stored in the List.
IndexOf	Returns the index of the first occurrence of the specified value in the List.
Insert	Inserts an element at the specified index.
Remove	Removes the first occurrence of the specified value.
RemoveAt	Removes the element at the specified index.
RemoveRange	Removes a specified number of elements starting at a specified index.
Sort	Sorts the List.
TrimExcess	Sets the Capacity of the List to the number of elements the List currently contains (Count).

Fig. 9.5 | Some methods and properties of class `List<T>`.

9.4.2 Ndryshimi dinamik i madhësisë së koleksionit List<T>

```
// Fig. 9.6: KoleksionMeList.cs
// Demonstrim i një koleksioni gjenerik (të përgjithshëm)
// të List<T>
using System;
using System.Collections.Generic;

class KoleksionMeList
{
    static void Main()
    {
        // krijoje një List të re të stringjeve
        var elementet = new List<string>();

        // shfaq Count dhe Capacity të Listës para shtimit të elementeve
        Console.WriteLine("Para shtimit të elementeve: " +
            $"Count (Sasia) = {elementet.Count}; Capacity (Kapaciteti) = {elementet.Capacity}");

        elementet.Add("e kuqe"); // shtoja (bashkangjitja në fund, append)
                                // një element Listës

        elementet.Insert(0, "e verdhë"); // fute vlerën tek indeksi 0

        // shfaq Count dhe Capacity të Listës pas shtimit të dy elementeve
        Console.WriteLine("Pas shtimit të dy elementeve në Listën elementet: " +
            $"Count = {elementet.Count}; Capacity = {elementet.Capacity}");

        // shfaq ngjyrat në Listë
        Console.Write(
            "\nShfaqe përmbajtjen e listës me cikël (unazë, loop) " +
            "të kontrolluar me vlerë vëzhguese (counter-controlled):");
        for (var i = 0; i < elementet.Count; i++)
        {
            Console.Write($" {elementet[i]}");
        }

        // shfaq ngjyrat duke përdorur foreach
        Console.Write("\nShfaqe përmbajtjen e listës me urdhër foreach:");
        foreach (var elementi in elementet)
        {
            Console.Write($" {elementi}");
        }

        elementet.Add("e gjelbër"); // shtojë "e gjelbër" në fund të Listës
        elementet.Add("e verdhë"); // shtojë "e verdhë" në fund të Listës

        // shfaq Count dhe Capacity të Listës pas shtimit të dy elementeve të tjera
        Console.WriteLine("\n\nPas shtimit të edhe dy elementeve të tjera në listën elementet: "
            + $"Count = {elementet.Count}; Capacity = {elementet.Capacity}");

        // shfaq Listën
        Console.Write("Lista me dy elemente të reja:");
        foreach (var elementi in elementet)
        {
            Console.Write($" {elementi}");
        }

        elementet.Remove("e verdhë"); // hiqe elementin e parë "e verdhë"
```

```

// shfaq Listën
Console.WriteLine("\nHiqe instancën e parë të 'e verdhë:");
foreach (var elementi in elementet)
{
    Console.WriteLine($" {elementi}");
}

elementet.RemoveAt(1); // hiqe elementin në indeksin 1

// shfaq Listën
Console.WriteLine("\nHiqe elementin e dytë në listë (e gjelbër:");
foreach (var elementi in elementet)
{
    Console.WriteLine($" {elementi}");
}

// shfaq Count dhe Capacity të Listës pas heqjes së dy elementeve
Console.WriteLine("\nPas heqjes së dy elementeve nga lista elementet: " +
    $"Count = {elementet.Count}; Capacity = {elementet.Capacity}");

// kontrollo se a është një vlerë në listë
Console.WriteLine("\n\"e kuqe\" " +
    $"{{(elementet.Contains(\"e kuqe\") ? string.Empty : \"nuk \"}}është në listë");

elementet.Add("e portokalltë"); // shtoj "e portokalltë" në fund të Listës
elementet.Add("e vjollctë"); // shtoj "e vjollctë" në fund të Listës
elementet.Add("e kaltër"); // shtoj "e kaltër" në fund të Listës

// shfaq Count dhe Capacity të Listës pas shtimit të tri elementeve
Console.WriteLine("\nPas shtimit të edhe tri elementeve në listën elementet: " +
    $"Count = {elementet.Count}; Capacity = {elementet.Capacity}");

// shfaq Listën
Console.WriteLine("Lista me tri elemente të reja:");
foreach (var elementi in elementet)
{
    Console.WriteLine($" {elementi}");
}
Console.WriteLine();
Console.ReadKey();
}
}

```

```

Para shtimit të elementeve: Count (Sasia) = 0; Capacity (Kapaciteti) = 0
Pas shtimit të dy elementeve në Listën elementet: Count = 2; Capacity = 4

Shfaq përmbajtjen e listës me cikël (unazë, loop) të kontrolluar me vlerë vëzhguese (counter-controlled): e verdhë e kuqe
Shfaq përmbajtjen e listës me urdhër foreach: e verdhë e kuqe

Pas shtimit të edhe dy elementeve të tjera në listën elementet: Count = 4; Capacity = 4
Lista me dy elemente të reja: e verdhë e kuqe e gjelbër e verdhë

Hiqe instancën e parë të 'e verdhë': e kuqe e gjelbër e verdhë
Hiqe elementin e dytë në listë (e gjelbër): e kuqe e verdhë
Pas heqjes së dy elementeve nga lista elementet: Count = 2; Capacity = 4

"e kuqe" është në listë

Pas shtimit të edhe tri elementeve në listën elementet: Count = 5; Capacity = 8
Lista me tri elemente të reja: e kuqe e verdhë e portokalltë e vjollctë e kaltër

```

9.5 Bërja e query-ve në koleksionin gjenerik List duke përdorur LINQ

- LINQ to Objects mund t'u bëjë query Listave.

```
// Fig. 9.7: LINQmeKoleksionList.cs
// LINQ to Objects (LINQ në Objekte)
// duke e përdorur një List<string>.
using System;
using System.Linq;
using System.Collections.Generic;

class LINQmeKoleksionList
{
    static void Main()
    {
        // populloje një Listë të stringjeve
        var elementet = new List<string>();
        elementet.Add("aQua"); // shtojë "aQua" në fund të Listës
        elementet.Add("RusT"); // shtojë "RusT" në fund të Listës
        elementet.Add("yELLow"); // shtojë "yELLow" në fund të Listës
        elementet.Add("rEd"); // shtojë "rEd" në fund të Listës

        // shfaqë Listën fillestare
        Console.WriteLine("Lista elementet përmban:");
        foreach (var elementi in elementet)
        {
            Console.WriteLine($"{elementi}");
        }

        Console.WriteLine(); // afishojë (nxirre në dalje) fundin e rreshtit

        // shndërroji në shkronja të mëdha (uppercase),
        // zgjedhi ato që nisin me R dhe sortoji
        var fillonMeR =
            from elementi in elementet
            let stringuMeShkronjaTeMedha = elementi.ToUpper()
            where stringuMeShkronjaTeMedha.StartsWith("R")
            orderby stringuMeShkronjaTeMedha
            select stringuMeShkronjaTeMedha;

        // shfaqë rezultatet e query-t (kërkesës/pyetësorit)
        Console.WriteLine("rezultatet e query-t fillonMeR:");
        foreach (var elementi in fillonMeR)
        {
            Console.WriteLine($"{elementi}");
        }

        Console.WriteLine(); // afishojë (nxirre në dalje) fundin e rreshtit

        elementet.Add("rUbY"); // shtojë "rUbY" në fund të Listës
        elementet.Add("SaFfRon"); // shtojë "SaFfRon" në fund të Listës

        // shfaqë Listën fillestare
        Console.WriteLine("Lista elementet përmban:");
        foreach (var elementi in elementet)
        {
            Console.WriteLine($"{elementi}");
        }
    }
}
```

```

Console.WriteLine(); // afishoje (nxirre në dalje) fundin e rreshtit

// shfaq rezultatat e query-t të përditësuar (updated)
Console.Write("rezultatet e query-t fillonMeR:");
foreach (var elementi in fillonMeR)
{
    Console.Write($" {elementi}");
}

Console.WriteLine(); // afishoje (nxirre në dalje) fundin e rreshtit
Console.ReadKey();
    }
}

```

```

Lista elementet përmban: aQua RuST yElLow rEd
rezultatet e query-t fillonMeR: RED RUST
Lista elementet përmban: aQua RuST yElLow rEd rUbY SaFfRon
rezultatet e query-t fillonMeR: RED RUBY RUST
_

```

9.5.1 Klauzola let

- Klauzola **let** e LINQ e krijon një variabël të re të rangut. Kjo është e dobishme nëse ju duhet ta ruani një rezultat të përkohshëm për ta përdorur më vonë në query-n LINQ.
- Metoda **StartsWith** e klasës **string** e përcakton se a fillon një **string** me **string**-un që i pasohet asaj si argument.

9.5.2 Ekzekutimi i shtyrë për më vonë

- Një query LINQ e përdor ekzekutimin e shtyrë për më vonë (deferred execution) – ajo ekzekutohet vetëm kur i qaseni rezultateve, jo kur e krijoni query-n.

9.5.3 Metodat e zgjerimit ToArray dhe ToList

- Mund të ketë raste kur dëshironi ta merrni menjëherë një koleksion të rezultateve. LINQ i ofron metodat e zgjerimit **ToArray** dhe **ToList** për këtë qëllim.

9.5.4 Inicializuesit e koleksioneve

- Inicializuesit e koleksioneve e ofrojnë një sintaksë të përshtatshme (të ngjashme me inicializuesit e vargjeve) për ta inicializuar një koleksion.

10. Klasat dhe objektet: vështrim më i thellë

10.2 Studim i rastit i klasës *Time*; hedhja e përjashtimeve

10.2.1 Deklarimi i klasës *Time1*

- Metodat **public**-e të një klase janë pjesë e shërbimeve publike apo ndërfaqja (interfejsi, interface) publik që ia ofron klasa klientëve të saj.
- Metodat dhe properti-t që i ndryshojnë vlerat e variablave **private** duhet të verifikojnë se vlerat e reja të synuara janë të vlefshme (valide).
- Metodat dhe properti-t e një klase mund të hedhin përjashtime (exceptions) për të treguar (indikuar) shënime jovalide (të pavlefshme).

```
// Fig. 10.1: Kohel.cs
// Deklarimi i klasës Kohel e mirëmban kohën në formatin 24-orësh.
using System; // namespace që e përmban ArgumentOutOfRangeException

public class Kohel
{
    public int Ora { get; set; } // 0 - 23
    public int Minuta { get; set; } // 0 - 59
    public int Sekonda { get; set; } // 0 - 59

    // caktoje një vlerë të re të kohës duke e përdorur kohën universale
    // hidhe një përjashtim nëse ora, minuta apo sekonda është jovalide
    public void CaktojeKohen(int ora, int minuta, int sekonda)
    {
        // valideje orën, minutën dhe sekondën
        if ((ora < 0 || ora > 23) || (minuta < 0 || minuta > 59) ||
            (sekonda < 0 || sekonda > 59))
        {
            throw new ArgumentOutOfRangeException();
        }

        Ora = ora;
        Minuta = minuta;
        Sekonda = sekonda;
    }

    // konvertoje (shndërroje) në string
    // në format të kohës universale (HH:MM:SS) apo shqip (00::MM::SS)
    public string NeStringUniversal() =>
        $"{Ora:D2}:{Minuta:D2}:{Sekonda:D2}";

    // konvertoje (shndërroje) në string
    // në format të kohës standarde (H:MM:SS AM apo PM)
    public override string ToString() =>
        $"{((Ora == 0 || Ora == 12) ? 12 : Ora % 12)}:" +
        $"{Minuta:D2}:{Sekonda:D2} {(Ora < 12 ? "AM" : "PM")}";
}
```


10.2.2 Përdorimi i klasës *Time1*

- Se cili përfaqësim (paraqitje, reprezentim) aktual i shënimeve përdoret brenda klasës nuk u intereson klientëve të klasës. Kjo ju lejon ta ndryshoni implementimin e klasës. Klientët do të mund t'i përdornin metodat dhe properti-t e njëjta **public**-e për t'i marrë rezultatet e njëjta pa qenë të vetëdijshëm (në dijeni) për këtë ndryshim.
- Klientët nuk janë as të vetëdijshëm (në dijeni) për, as të përfshirë në, implementimin e një klase. Klientëve përgjithësisht u intereson se *çka* bën klasa por jo se *si* e bën klasa atë.

```
// Fig. 10.2: TestKohe1.cs
// Objekti Kohe1 i përdorur në një aplikacion.
using System;

class TestKohe1
{
    static void Main()
    {
        // krijoje dhe inicializojë një objekt Koha1
        var koha = new Kohe1(); // e thirr konstruktorin Koha1

        // shfaq paraqitjet string të kohës
        Console.WriteLine(
            $"Koha fillestare universale është: {koha.NeStringUniversal()}");
        Console.WriteLine(
            $"Koha fillestare standarde është: {koha.ToString()}");
        Console.WriteLine(); // shfaqë një rresht të zbrazët

        // ndryshojë kohën dhe shfaqë kohën e përditësuar
        koha.CaktojeKohen(13, 27, 6);
        Console.WriteLine(
            $"Koha universale pas CaktojeKohen është: {koha.NeStringUniversal()}");
        Console.WriteLine(
            $"Koha standarde pas CaktojeKohen është: {koha.ToString()}");
        Console.WriteLine(); // shfaqë një rresht të zbrazët

        // tento ta caktosh kohën me vlera jovalide
        try
        {
            koha.CaktojeKohen(99, 99, 99);
        }
        catch (ArgumentOutOfRangeException ex)
        {
            Console.WriteLine(ex.Message + "\n");
        }

        // shfaqë kohën pas tentimit të caktohen vlera jovalide
        Console.WriteLine("Pas tentimit të cilësimeve jovalide:");
        Console.WriteLine($"Koha universale: {koha.NeStringUniversal()}");
        Console.WriteLine($"Koha standarde: {koha.ToString()}");
        Console.ReadKey();
    }
}
```

```
Koha fillestare universale është: 00:00:00
Koha fillestare standarde është: 12:00:00 AM

Koha universale pas CaktojeKohen është: 13:27:06
Koha standarde pas CaktojeKohen është: 1:27:06 PM

Specified argument was out of the range of valid values.

Pas tentimit të cilësimeve jovalide:
Koha universale: 13:27:06
Koha standarde: 1:27:06 PM
```

10.3 Kontrollimi i qasjes tek anëtarët

- Modifikuesit (ndryshuesit) e qasjes **public** dhe **private** e kontrollojnë qasjen te variablat, metodat dhe proprietit e klasës. Variablat, metodat dhe proprieti-t **private** të klasës nuk janë të qasshme drejtpërdrejt (direkt) për klientët e klasës.
- Nëse një klient tenton (përpiqet) t'i përdorë anëtarët privatë të një klase tjetër, kompajleri gjeneron (prodhon) mesazhe të gabimit që deklarojnë se këta anëtarë privatë nuk janë të qasshëm.

```
// Fig. 10.1: Kohe1.cs
// Deklarimi i klasës Kohe1 e mirëmban kohën në formatin 24-orësh.
using System; // namespace që e përmban ArgumentOutOfRangeException

public class Kohe1
{
    private int ora; // 0 - 23
    private int minuta; // 0 - 59
    private int sekonda; // 0 - 59

    // caktoje një vlerë të re të kohës duke e përdorur kohën universale
    // hidhe një përjashtim nëse ora, minuta apo sekonda është jovalidë
    public void CaktojeKohen(int o, int m, int s)
    {
        // validojë orën, minutën dhe sekondën
        if ((o >= 0 && o < 24) && (m >= 0 && m < 60) &&
            (s >= 0 && s < 60))
        {
            ora = o;
            minuta = m;
            sekonda = s;
        } // përfundoje if-in
        else
            throw new ArgumentOutOfRangeException();
    } // përfundoje metodën SetTime

    // konvertoje (shndërroje) në string
    // në format të kohës universale (HH:MM:SS) apo shqip (00::MM::SS)
    public string NeStringUniversal()
    {
        return string.Format("{0:D2}:{1:D2}:{2:D2}",
            ora, minuta, sekonda);
    } // përfundoje metodën NeStringUniversal

    // konvertoje (shndërroje) në string
    // në format të kohës standarde (H:MM:SS AM apo PM)
    public override string ToString()
    {
        return string.Format("{0}:{1:D2}:{2:D2} {3}",
            ((ora == 0 || ora == 12) ? 12 : ora % 12),
            minuta, sekonda, (ora < 12 ? "AM" : "PM"));
    } // përfundoje metodën ToString
} // përfundoje klasën Kohe1
```

```
// Fig. 10.3: TestIQasjesSeAnetareve.cs
// Anëtarët privatë të klasës Koha1 nuk janë të qasshëm jashtë klasës.
class TestIQasjesSeAnetareve
{
    static void Main()
    {
        var koha = new Kohe1(); // krijoje dhe inicializojë objektin Koha1

        koha.ora = 7; // gabim: ora ka qasje private në Koha1
        koha.minuta = 15; // gabim: minuta ka qasje private në Koha1
        koha.sekonda = 30; // gabim: sekonda ka qasje private në Koha1
    }
}
```

Error List		
Entire Solution		
<div> ✖ 3 Errors ⚠ 0 Warnings ℹ 0 of 1 Message </div>		
	Code	Description
✖	CS0122	'Kohe1.ora' is inaccessible due to its protection level
✖	CS0122	'Kohe1.minuta' is inaccessible due to its protection level
✖	CS0122	'Kohe1.sekonda' is inaccessible due to its protection level

10.4 Referimi tek anëtarët e objektit aktual me referencën this

- Çdo objekt mund t'i qaset një reference tek vetja me fjalën kyçe (keyword) **this** (ky apo kjo). Kur një metodë jo-statike thirret për një objekt të caktuar, trupi i metodës e përdor në mënyrë implicite fjalën kyçe **this** për t'iu referuar variablave të instancës, metodave të tjera dhe properti-ve të objektit.
- Nëse një metodë e përmban një variabël lokale me emrin e njëjtë si një fushë, ajo metodë do t'i referohet variablës lokale e jo fushës. Mirëpo, një metodë jo-statike (non-**static**) mund ta përdorë referencën **this** për t'iu referuar në mënyrë eksplicite një variable të fshehur të instancës.
- Shmangiuni emrave të parametrave të metodave apo emrave të variablave lokale që bien ndesh (në konflikt) me emrat e fushave. Kjo ndihmon të parandalohen bug-a (defekte) subtile, që gjenden vështirë.

```
// Fig. 10.4: TestIThis.cs
// this e përdorur në mënyrë implicite dhe eksplite
// për t'iu referuar anëtarëve të një objekti.
using System;

class TestIThis
{
    static void Main()
    {
        var koha = new KoheETHjeshte(15, 30, 19);
        Console.WriteLine(koha.NdertojeStringun());
        Console.ReadKey();
    }
}

// klasa KoheETHjeshte e demonstroi referencën "this"
public class KoheETHjeshte
{
    private int ora; // 0-23
    private int minuta; // 0-59
    private int sekonda; // 0-59

    // nëse konstruktori i përdor emrat e parametrave
    // që janë identikë me emrat e variablave të instancave,
    // kërkohet referenca "this" për të bërë dallim ndërmjet emrave
    public KoheETHjeshte(int ora, int minuta, int sekonda)
    {
        this.ora = ora; // caktojë instancën e variablës ora të këtij ("this") objekti
        this.minuta = minuta; // caktojë minutën e këtij ("this") objekti
        this.sekonda = sekonda; // caktojë sekondën e këtij ("this") objekti
    }

    // përdore "this"-in eksplisit dhe implicit për ta thirrur NeStringUniversal
    public string NdertojeStringun() =>
        $"{this.NeStringUniversal()}", 24}: {this.NeStringUniversal()}" +
        $"{\n{"NeStringUniversal()", 24}: {NeStringUniversal()}}";

    // konvertoje (shndërroje) në string në formatin e kohës universale (HH:MM:SS);
    // "this" nuk kërkohet këtu për t'iu qasur variablave të instancës,
    // sepse metoda nuk ka variabla lokale me emra të njëjtë
    // si variablat e instancës
    public string NeStringUniversal() =>
        $"{this.ora:D2}:{this.minuta:D2}:{this.sekonda:D2}";
}
```

```
this.NeStringUniversal(): 15:30:19
NeStringUniversal(): 15:30:19
```

10.5 Konstruktoret e mbingarkuar

- Për t'i mbingarkuar konstruktoret, ofroni shumë deklarime të konstruktorëve me nënshkrime (signatures) të ndryshme.
- Nëse pas ballinës (header-it) të konstruktorit e shkruajmë inicializuesin e konstruktorit : **this (args)** kjo e thirr konstruktorin e mbingarkuar përkatës (që përputhet, matching) në klasën e njëjtë.
- Inicializuesit e konstruktorit janë mënyrë e përhapur për ta ripërdorur kodin e inicializimit të ofruar prej një nga konstruktoret e klasës në vend se të definohet kod i ngjashëm në trupin e një konstruktori tjetër.
- Kur një objekt i një klase ka referencë në një objekt tjetër të klasës së njëjtë, objekti i parë mund t'u qaset të gjitha shënimeve dhe metodave të objektit të dytë (përfshirë edhe ato që janë **private**).
- Kur e implementoni një metodë të klasës, përdorni properti-t e klasës për t'iu qasur shënimeve **private** të klasës. Kjo e thjeshton mirëmbajtjen e kodit dhe i zvogëlon gjasat e gabimeve.
- Konstruktori **ArgumentOutOfRangeException** me tri argumente ju lejon ta specifikoni emrin e elementit (item-it) që është jashtë rangut, vlerën që ishte jashtë rangut dhe një mesazh të gabimit.

```
// Fig. 10.5: Kohe2.cs
// Deklarimi i klasës Kohe2 me konstruktorë të mbingarkuar.

using System; // për klasën ArgumentOutOfRangeException

public class Kohe2
{
    private int ora; // 0 - 23
    private int minuta; // 0 - 59
    private int sekonda; // 0 - 59

    // konstruktori mund të thirret me zero, një, dy apo tri argumente
    public Kohe2(int ora = 0, int minuta = 0, int sekonda = 0)
    {
        CaktojeKohen(ora, minuta, sekonda); // thirrë CaktojeKohen për ta validuar kohën
    }

    // konstruktori Kohe2: një objekt tjetër Kohe2 i dhënë si argument
    public Kohe2(Kohe2 koha)
        : this(koha.Ora, koha.Minuta, koha.Sekonda) { }

    // caktoje një vlerë të re të kohës duke e përdorur kohën universale;
    // vlerat jovalide shkaktajnë që aksesorët (qasësit) set
    // të properti-ve të hedhin përjashtime (throw exceptions)
    public void CaktojeKohen(int ora, int minuta, int sekonda)
    {
        Ora = ora; // caktoje propertin Ora
        Minuta = minuta; // caktoje propertin Minuta
        Sekonda = sekonda; // caktoje propertin Sekonda
    }

    // properti që e merr (gets) dhe e cakton (sets) orën
    public int Ora
    {
        get
        {
            return ora;
        }
        set
        {
            if (value < 0 || value > 23)
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(Ora)} duhet të jetë 0-23");
            }

            ora = value;
        }
    }
}
```

```
// properti që e merr (gets) dhe e cakton (sets) minutën
public int Minuta
{
    get
    {
        return minuta;
    }
    set
    {
        if (value < 0 || value > 59)
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(Minuta)} duhet të jetë 0-59");
        }

        minuta = value;
    }
}

// properti që e merr (gets) dhe e cakton (sets) sekondën
public int Sekonda
{
    get
    {
        return sekonda;
    }
    set
    {
        if (value < 0 || value > 59)
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(Sekonda)} duhet të jetë 0-59");
        }

        sekonda = value;
    }
}

// konvertoje në string në format të kohës universale (HH:MM:SS)
public string NeStringUniversal() =>
    $"{Ora:D2}:{Minuta:D2}:{Sekonda:D2}";

// konvertoje në string në format të kohës standarde (H:MM:SS AM or PM)
public override string ToString() =>
    $"{((Ora == 0 || Ora == 12) ? 12 : Ora % 12)}:" +
    $"{Minuta:D2}:{Sekonda:D2} {(Ora < 12 ? "AM" : "PM")}";
}
```



```

// Fig. 10.6: TestIKohe2.cs
// Konstruktoret e mbingarkuar
// të përdorur për të inicializuar objekte Kohe2
using System;

public class TestIKohe2
{
    static void Main()
    {
        var k1 = new Kohe2(); // 00:00:00
        var k2 = new Kohe2(2); // 02:00:00
        var k3 = new Kohe2(21, 34); // 21:34:00
        var k4 = new Kohe2(12, 25, 42); // 12:25:42
        var k5 = new Kohe2(k4); // 12:25:42

        Console.WriteLine("Konstruktuar me:\n");
        Console.WriteLine("k1: të gjitha argumentet të paracaktuara (defaulted)");
        Console.WriteLine($" {k1.NeStringUniversal()}"); // 00:00:00
        Console.WriteLine($" {k1.ToString()}\n"); // 12:00:00 AM

        Console.WriteLine(
            "k2: ora e specifikuar; minuta dhe sekonda të paracaktuara (defaulted)");
        Console.WriteLine($" {k2.NeStringUniversal()}"); // 02:00:00
        Console.WriteLine($" {k2.ToString()}\n"); // 2:00:00 AM

        Console.WriteLine(
            "k3: ora dhe minuta të specifikuar; sekonda e paracaktuar (defaulted)");
        Console.WriteLine($" {k3.NeStringUniversal()}"); // 21:34:00
        Console.WriteLine($" {k3.ToString()}\n"); // 9:34:00 PM

        Console.WriteLine("k4: ora, minuta dhe sekonda të specifikuar");
        Console.WriteLine($" {k4.NeStringUniversal()}"); // 12:25:42
        Console.WriteLine($" {k4.ToString()}\n"); // 12:25:42 PM

        Console.WriteLine("k5: objekti k4 i klasës Time2 i specifikuar");
        Console.WriteLine($" {k5.NeStringUniversal()}"); // 12:25:42
        Console.WriteLine($" {k5.ToString()}"); // 12:25:42 PM

        // tento ta inicializosh k6 me vlera jovalide
        try
        {
            var k6 = new Kohe2(27, 74, 99); // vlera jovalide
        }
        catch (ArgumentOutOfRangeException ex)
        {
            Console.WriteLine("\nPërrjashtim (Exception) gjatë inicializimit të k6:");
            Console.WriteLine(ex.Message);
        }
        Console.ReadKey();
    }
}

```

```
Konstruktuar me:

k1: të gjitha argumentet të paracaktuara (defaulted)
    00:00:00
    12:00:00 AM

k2: ora e specifikuar; minuta dhe sekonda të paracaktuara (defaulted)
    02:00:00
    2:00:00 AM

k3: ora dhe minuta të specifikuara; sekonda e paracaktuar (defaulted)
    21:34:00
    9:34:00 PM

k4: ora, minuta dhe sekonda të specifikuara
    12:25:42
    12:25:42 PM

k5: objekti k4 i klasës Time2 i specifikuar
    12:25:42
    12:25:42 PM

Përjashtim (Exception) gjatë inicializimit të k6:
Ora duhet të jetë 0-23
Parameter name: value
Actual value was 27.
```

10.6 Konstruktoret default (të paracaktuar) dhe pa parametra

- Çdo klasë duhet ta ketë të paktën një konstruktor. Nëse nuk ka konstruktorë në deklarimin e një klase, kompajleri e krijon një konstruktor default për klasën.
- Kompajleri nuk do të krijojë konstruktor default për një klasë që e deklaron në mënyrë eksplicite të paktën një konstruktor. Në këtë rast, nëse dëshironi të jeni në gjendje ta thirrni konstruktorin pa argumente, duhet ta deklaroni një konstruktor pa parametra.

10.7 Kompozimi (Composition)

- Një klasë mund të ketë si anëtarë objekte të tipeve vlerë apo referenca të objekteve të klasave të tjera. Një mundësi (aftësi) e tillë quhet kompozim dhe nganjëherë e quajmë (i referohemi si) relacion (marrëdhënie) **has-a (e ka një)**.

```
// Fig. 10.7: Datë.cs
// Deklarimi i klasës Datë
using System;

public class Datë
{
    private int muaji; // 1-12
    private int dita; // 1-31 based on month
    public int Viti { get; private set; } // properti Viti e implementuar automatikisht

    // konstruktori: përdore properti-n Muaji për ta konfirmuar vlerën e duhur për muaj;
    // përdore properti-n Dita për ta konfirmuar vlerën e duhur për ditë;
    public Datë(int muaji, int dita, int viti)
    {
        Muaji = muaji; // e validon muajin
        Viti = viti; // do të mund ta validonte vitin
        Dita = dita; // e validon ditën
        Console.WriteLine($"Konstruktori i objektit Datë për datën {this}");
    }

    // properti që e merr (gets) dhe e cakton (sets) muajin
    public int Muaji
    {
        get
        {
            return muaji;
        }
        private set // bëje shkrimin të paqasshëm jashtë klasës
        {
            if (value <= 0 || value > 12) // validoje muajin
            {
                throw new ArgumentOutOfRangeException(
                    nameof(value), value, $"{nameof(Muaji)} duhet të jetë 1-12");
            }

            muaji = value;
        }
    }
}
```

```

// properti që e merr (gets) dhe e cakton (sets) ditën
public int Dita
{
    get
    {
        return dita;
    }
    private set // bëje shkrimin të paqasshëm jashtë klasës
    {
        int[] ditePerMuaj =
            {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

        // kontrolloje a është dita në rangun për muaj
        if (value <= 0 || value > ditePerMuaj[Muaji])
        {
            throw new ArgumentOutOfRangeException(nameof(value), value,
                $"{nameof(Dita)} jashtë rangut për muajin/vitin aktual");
        }
        // kontrollo për vitin e brishtë (leap year)
        if (Muaji == 2 && value == 29 &&
            !(Viti % 400 == 0 || (Viti % 4 == 0 && Viti % 100 != 0)))
        {
            throw new ArgumentOutOfRangeException(nameof(value), value,
                $"{nameof(Dita)} jashtë rangut për muajin/vitin aktual");
        }

        dita = value;
    }
}

// ktheje një string të formës muaji/dita/viti
public override string ToString() => $"{Muaji}/{Dita}/{Viti}";
}

```

```

// Fig. 10.8: Punonjes.cs
// Klasa Punonjes me referenca në objekte të tjera
public class Punonjes
{
    public string Emri { get; }
    public string Mbiemri { get; }
    public Datë Datelindja { get; }
    public Datë DataEPunesimit { get; }

    // konstruktori për ta inicializuar emrin, datëlindjen dhe datën e punësimit
    public Punonjes(string emri, string mbiemri,
        Datë datelindja, Datë dataEPunesimit)
    {
        Emri = emri;
        Mbiemri = mbiemri;
        Datelindja = datelindja;
        DataEPunesimit = dataEPunesimit;
    }

    // konvertoje (shndërroje) objektin Punonjës në format string
    public override string ToString() => $"{Mbiemri}, {Emri} " +
        $"{I punësuar më: {DataEPunesimit} Datëlindja: {Datelindja}";
}

```

```
// Fig. 10.9: TestIPunonjes.cs
// Demonstrim i kompozimit.
using System;

class TestIPunonjes
{
    static void Main()
    {
        var datelindja = new Datë(7, 24, 1949);
        var dataEPunesimit = new Datë(3, 12, 1988);
        var punonjesi = new Punonjes("Bob", "Blue", datelindja, dataEPunesimit);

        Console.WriteLine(punonjesi);
        Console.ReadKey();
    }
}
```

```
Konstruktori i objektit Datë për datën 7/24/1949
Konstruktori i objektit Datë për datën 3/12/1988
Blue, Bob I punësuar më: 3/12/1988 Datëlindja: 7/24/1949
```

10.8 Garbage collection (mbledhja e mbeturinave) dhe destrukturët (shkatërruesit)

- Secili objekt që e krijoni përdor resurse të ndryshme të sistemit, siç është memoria. CLR (Common Language Runtime, softueri i dizajnuar që ta përkrahë ekzekutimin e programeve që është i përbashkët për të gjitha gjuhët .NET) e kryen menaxhimin automatik të memories (kujtesës) duke e përdorur një garbage collector (mbledhës të mbeturinave) për ta rikthyer memorien e zënë nga objekte që nuk janë më në përdorim.
- Destruktori thirret nga garbage collector për ta kryer mirëmbajtjen e terminimit (përfundimit të “jetës”) në një objekt para se garbage collector-i ta rimarrë memorien e objektit.
- Memory leaks (rrjedhjet e memories), që janë të zakonshme në gjuhë të tjera si C dhe C++ (për shkak se memoria nuk rimirret automatikisht në ato gjuhë), kanë më pak gjasa të ndodhin në C#.
- Një problem me garbage collector-in është se nuk garantohet se ai do t’i kryejë detyrat e tij në një kohë të caktuar. Prandaj, garbage collector-i mund ta thërrasë destrukturin në çdo kohë pasi objekti të bëhet i lejueshëm (eligible) për shkatërrim, duke e bërë të paqartë se kur, apo a do të thirret destruktori.

10.9 Anëtarët *static*-ë të klasës

- Një variabël **static**-e paraqet informacion për tërë klasën – të gjitha objektet e klasës e bashkë-përdorin (e ndajnë, share) variablën.
- Skopi i një variable **static**-e është trupi i klasës së saj. Anëtarët **public**-ë **static**-ë të klasës mund të qasen duke e kualifikuar emrin e anëtarit me emrin e klasës dhe operatorin e qasjes së anëtarit (.).
- Anëtarët statikë të klasës ekzistojnë edhe kur nuk ekzistojnë objekte të klasës – ata janë të disponueshëm posa të ngarkohet klasa në memorie në kohën e ekzekutimit.
- Objektet string në C# janë të pandryshueshme – ato nuk mund të ndryshohen pasi të krijohen. Prandaj është e sigurt të keni shumë referenca në një objekt **string**.
- Një metodë e deklaruar **static**-e nuk mund t’u qaset drejtpërdrejt anëtarëve jo-**static**-ë të klasës, sepse një metodë **static**-e mund të thirret madje edhe kur nuk ekzistojnë objekte të klasës. Për të njëjtën arsye, referenca **this** nuk mund të përdoret në metodë **static**-e.

```
// Fig. 10.10: Punonjes.cs
// properti static-e e përdorur për ta mirëmbajtur
// numrin e objekteve Punonjës që janë krijuar.
using System;

class Punonjes
{
    public static int Numeruesi { get; private set; } // objekte në memorie

    public string Emri { get; }
    public string Mbiemri { get; }

    // inicializojë punonjësën, shtoja 1 Numeruesi-t statik dhe
    // shfaqë stringun që tregon se është thirrur konstruktori
    public Punonjes(string emri, string mbiemri)
    {
        Emri = emri;
        Mbiemri = mbiemri;
        ++Numeruesi; // rrite numeruesin statik të punonjësve
        Console.WriteLine("Konstruktori Punonjes: " +
            $"{Emri} {Mbiemri}; Numëruesi = {Numeruesi}");
    }
}
```

```
// Fig. 10.11: TestIPunonjes.cs
// demonstrim i anëtarit statik.
using System;

public class TestIPunonjes
{
    static void Main()
    {
        // trego se Numeruesi është 0 para krijimit të objekteve Punonjës
        Console.WriteLine(
            $"Numri i objekteve Punonjës para instancimit (instantiation): {Punonjes.Numeruesi}");

        // krijoji dy Punonjes; Numeruesi duhet të bëhet 2
        var p1 = new Punonjes("Susan", "Baker");
        var p2 = new Punonjes("Bob", "Blue");

        // trego se Numeruesi është 2 pas krijimit të dy objekteve Punonjes
        Console.WriteLine(
            $"\\n Punonjës pas instancimit: {Punonjes.Numeruesi}");

        // merri emrat e Punonjësve
        Console.WriteLine($"\\n Punonjësi 1: {p1.Emri} {p1.Mbiemri}");
        Console.WriteLine($"Punonjësi 2: {p2.Emri} {p2.Mbiemri}");

        // në këtë shembull, është vetëm nga një referencë te secili Punonjes,
        // kështu që urdhrat vijues shkarkojnë që CLR ta shenjojë secilin
        // objekt Punonjes si të gatshëm për garbage collection (mbledhje të mbeturinave)
        p1 = null; // shenjoje objektin e referencuar nga p1 si asi që nuk duhet më
        p2 = null; // shenjoje objektin e referencuar nga p2 si asi që nuk duhet më
        Console.ReadKey();
    }
}
```

```
Numri i objekteve Punonjës para instancimit (instantiation): 0
Konstruktori Punonjes: Susan Baker; Numëruesi = 1
Konstruktori Punonjes: Bob Blue; Numëruesi = 2

Punonjës pas instancimit: 2

Punonjësi 1: Susan Baker
Punonjësi 2: Bob Blue
```

10.10 Variablat e instancës që janë *readonly* (vetëm për lexim)

- Parimi i privilegjit më të pakët (më të vogël, The principle of least privilege) është themeltar për inxhinierim të mirë të softuerit. Në kontekstin e një aplikacioni, parimi thotë se kodit duhet t'i jepet vetëm sasia e privilegjit dhe e qasjes që nevojitet për ta përmbushur detyrën e tij të përcaktuar, por jo më shumë.
- Çdo përpjekje për ta ndryshuar një variabël të instancës që është **readonly** pasi të jetë konstruktuar objekti i saj është gabim.
- Megjithëse variablat e instancës që janë readonly mund të inicializohen kur deklarohen, kjo nuk kërkohet. Një variabël readonly mund të inicializohet nga secili konstruktor i klasës.
- Anëtarëve që janë deklaruar si **const** duhet t'u caktohet vlera në kohën e kompajlimit (compile time). Anëtarët konstantë me vlera që nuk mund të përcaktohen në kohën e kompajlimit duhet të deklarohen me fjalën kyçe **readonly**, ashtu që të mund të inicializohen në kohën e ekzekutimit.
- Kur një properti e implementuar automatikisht ka vetëm qasës (accessor) **get**, properti mund të përdoret vetëm për ta lexuar vlerën, kështu që kompajleri e deklaron në mënyrë implicite variablën përkatëse private të instancës si **readonly**.

10.11 Class View (Pamja e klasave) dhe Object Browser (Shfletuesi i Objekteve)

- **(10.11.1 Përdorimi i dritares *Class View*) Class View** (Pamja e klasave) i shfaq fushat, metodat dhe propriet për të gjitha klasat në një projekt. Pamja e ndjek një strukturë hierarkike, duke e pozicionuar emrin e projektit si rrënjë dhe duke e përfshirë një seri të nyjeve (nodes) që i paraqesin klasat, fushat, metodat dhe propriet në projekt.
- **(10.11.2 Përdorimi i *Object Browser*) Object Browser** (Shfletuesi i objekteve) i liston të gjitha klasat e Framework Class Library (Librarisë së Klasave të Kornizës .NET). **Object Browser** mund të jetë mekanizëm i shpejtë për të mësuar për një klasë apo një metodë të një klase.

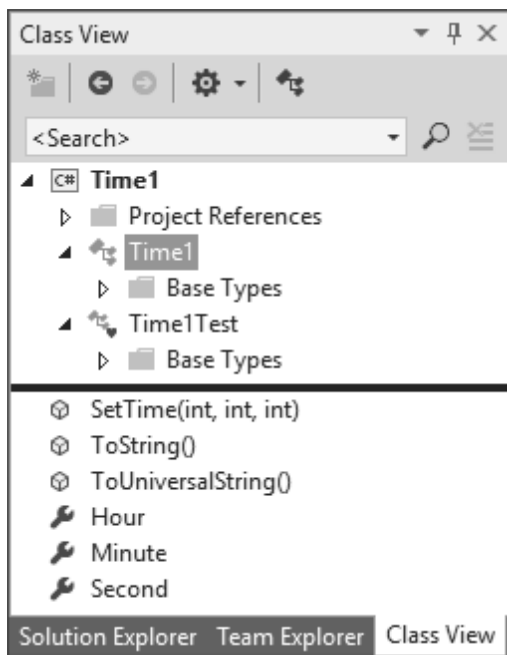


Fig. 10.12 | Class View of class Time1 (Fig. 10.1) and class Time1Test (Fig. 10.2).

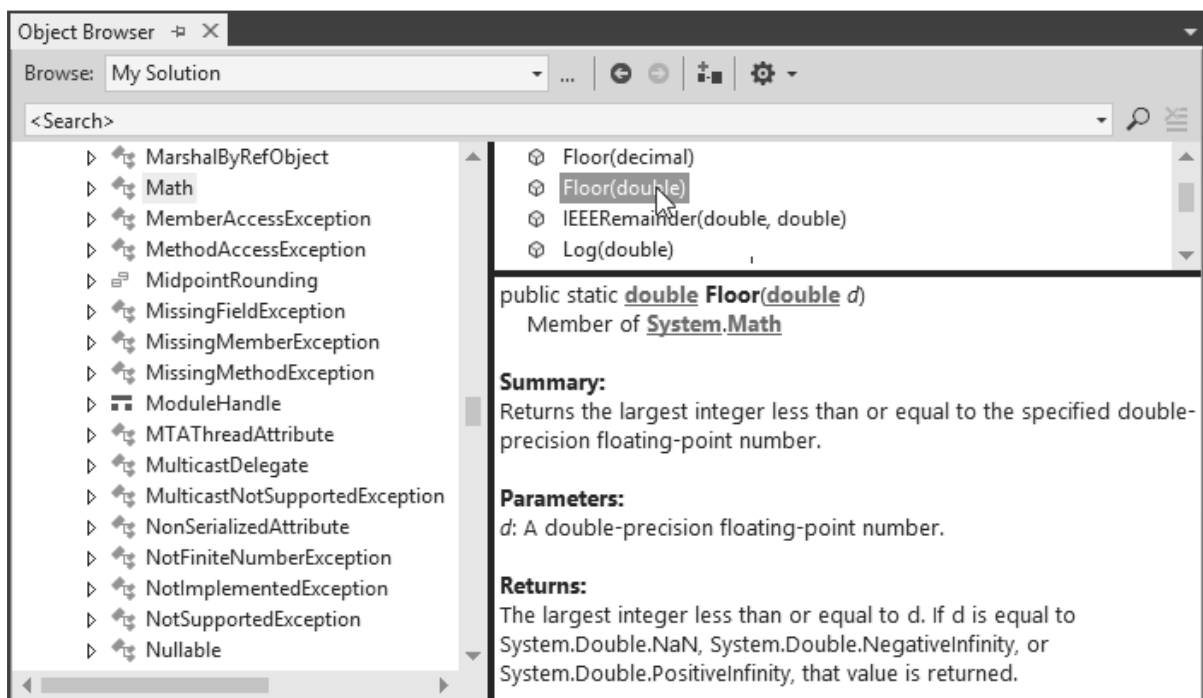


Fig. 10.13 | Object Browser for class Math.

10.12 Inicializuesit e objekteve

- Inicializuesit e objekteve (object initializers) ju lejojnë ta krijoni një objekt dhe t'i inicializoni propriet **public**-e të tij (dhe variablat **public**-e të instancës, nëse ka) në urdhrin e njëjtë.
- Një listë e inicializuesit të objektit (object-initializer list) është listë e ndarë me presje në kllapa gjarpërore ({}) e properti-ve (dhe dhe variabla **public**-e të instancës, nëse ka) dhe e vlerave të tyre.
- Secili emër i propriet dhe i variablës së instancës mund të shfaqet vetëm një herë në listën e inicializuesit të objektit.
- Një inicializues i objektit së pari e thirr konstruktorin e klasës, pastaj e cakton në radhën në të cilën shfaqen vlerën e secilës properti dhe variabël të specifikuar në listën e inicializuesit të objektit.

10.13 Mbingarkimi i operatorëve; Prezentimi i *struct*

- Notacioni i thirrjes së metodave (method-call notation) është i papërshtatshëm për lloje të caktuara të klasave, veçanërisht për klasat matematike. Nganjëherë është e përshtatshme të përdoren operatorët e integruar të C# për t'i specifikuar manipulimet e objekteve.

10.13.1 Krijimi i tipeve vlerë me *struct*

- Mund t'i definoni tipet tuaja vlerë si **struct**-ë.
- Tipet e thjeshta të C# si `int` dhe `double` janë në fakt pseudonime (aliases) për tipe **struct**.
- Microsoft e rekomandon përdorimin e klasave për shumicën e tipeve të reja, por rekomandon **struct** nëse: tipi e paraqet një vlerë të vetme dhe madhësia e një objekti është 16 bajtë apo më e vogël.

10.13.2 Tipi vlerë *ComplexNumber* (NumërKompleks)

- Fjala kyçe **operator**, e pasuar nga një operator, tregon se një metodë e mbingarkon operatorin e specifikuar. Metodat që i mbingarkojnë operatorët binarë duhet të deklarohen **static**-e dhe duhet t'i marrin dy argumente. Argumenti i parë është operandi i majtë dhe i dyti është operandi i djathtë.
- Mbingarkoni operatorët për ta performuar (kryer) funksionin e njëjtë apo funksione të ngjashme në objekte të klasës siç i performojnë operatorët në objektet e tipeve të thjeshta. Shmangiuni përdorimit jointuitiv të operatorëve.

```
// Fig. 10.14: NumerKompleks.cs
// Tip vlerë që i mbingarkon operatorët për mbledhje, zbritje
// dhe shumëzim të numrave kompleksë.
using System;

public struct NumerKompleks
{
    // properti-t vetëm për lexim (read-only)
    // që i marrin komponentat reale dhe imagjinare
    public double PjesaReale { get; }
    public double PjesaImagjinare { get; }

    // constructor
    public NumerKompleks(double pjesaReale, double pjesaImagjinare)
    {
        PjesaReale = pjesaReale;
        PjesaImagjinare = pjesaImagjinare;
    }

    // ktheje paraqitjen string të objektit NumerKompleks
    public override string ToString() =>
        $"({PjesaReale} {(PjesaImagjinare < 0 ? "-" : "+" )} {Math.Abs(PjesaImagjinare)})i";

    // mbingarkoje operatorin e mbledhjes
    public static NumerKompleks operator +(NumerKompleks x, NumerKompleks y)
    {
        return new NumerKompleks(x.PjesaReale + y.PjesaReale,
            x.PjesaImagjinare + y.PjesaImagjinare);
    }

    // mbingarkoje operatorin e zbritjes
    public static NumerKompleks operator -(NumerKompleks x, NumerKompleks y)
    {
        return new NumerKompleks(x.PjesaReale - y.PjesaReale,
            x.PjesaImagjinare - y.PjesaImagjinare);
    }

    // mbingarkoje operatorin e shumëzimit
    public static NumerKompleks operator *(NumerKompleks x, NumerKompleks y)
    {
        return new NumerKompleks(
            x.PjesaReale * y.PjesaReale - x.PjesaImagjinare * y.PjesaImagjinare,
            x.PjesaReale * y.PjesaImagjinare + y.PjesaReale * x.PjesaImagjinare);
    }
}
```

```
// Fig. 10.15: TestKompleks.cs
// Mbingarkimi i operatorëve për numra kompleksë.
using System;

class TestKompleks
{
    static void Main()
    {
        // kërko nga përdoruesi ta shkruajë numrin e parë kompleks
        Console.WriteLine("Shkruaje pjesën reale të numrit kompleks x: ");
        double pjesaReale = double.Parse(Console.ReadLine());
        Console.WriteLine("Shkruaje pjesën imagjinare të numrit kompleks x: ");
        double pjesaImagjinare = double.Parse(Console.ReadLine());
        var x = new NumerKompleks(pjesaReale, pjesaImagjinare);

        // kërko nga përdoruesi ta shkruajë numrin e dytë kompleks
        Console.WriteLine("\nShkruaje pjesën reale të numrit kompleks y: ");
        pjesaReale = double.Parse(Console.ReadLine());
        Console.WriteLine("Shkruaje pjesën imagjinare të numrit kompleks y: ");
        pjesaImagjinare = double.Parse(Console.ReadLine());
        var y = new NumerKompleks(pjesaReale, pjesaImagjinare);

        // shfaq rezultatat e llogaritjeve (kalkulimeve) me x dhe y
        Console.WriteLine();
        Console.WriteLine($"{x} + {y} = {x + y}");
        Console.WriteLine($"{x} - {y} = {x - y}");
        Console.WriteLine($"{x} * {y} = {x * y}");
        Console.ReadKey();
    }
}
```

```
Shkruaje pjesën reale të numrit kompleks x: 1
Shkruaje pjesën imagjinare të numrit kompleks x: 2

Shkruaje pjesën reale të numrit kompleks y: 2
Shkruaje pjesën imagjinare të numrit kompleks y: 1

(1 + 2i) + (2 + 1i) = (3 + 3i)
(1 + 2i) - (2 + 1i) = (-1 + 1i)
(1 + 2i) * (2 + 1i) = (0 + 5i)
```

10.14 Studimi i rastit Klasa Kohë: metodat zgjeruese (Extension Methods)

- Metodat zgjeruese i shtojnë funksionalitet një tipi pa e ndryshuar kodin burimor të tipeve.
- Kapacitetet (aftësitë) LINQ implementohen si metoda shpresë.
- Kur një parametër të objektit që është parametri i parë në ballinën e metodës e paraprijmë me fjalën kyçe **this** kjo tregon se metoda e zgjeron një tip ekzistues. Kompajleri e përdor këtë informacion për të shtuar kod në programin e kompajluar që ua mundëson metodave zgjeruese për të punuar me tipet ekzistuese.
- Tipi i parametrin të parë të metodës së zgjerimit e specifikon tipin që po e zgjeroni.
- Metodat zgjeruese duhet të definohen si metoda **static**-e në klasë **static**-e.
- Kompajleri e pason në mënyrë implicite objektin që përdoret për ta thirrur metodën si argument të parë të metodës zgjeruese. Kjo ju lejon ta thirrni një metodë zgjeruese sikur të ishte metodë e instancës e klasës së zgjeruar.
- *IntelliSense* i shfaq metodat zgjeruese me metodat e instancës së klasës së zgjeruar dhe i identifikon ato me ikonë të veçantë.
- Thirrjet kaskadë (ujëvarë, cascaded) të metodave thirren nga e majta në të djathtë.
- Emri plotësisht i kualifikuar i një metode zgjeruese është emri i klasës në të cilën është definuar metoda zgjeruese, pasuar nga emri i metodës dhe lista e argumenteve të saj. Kur e përdorni emrin plotësisht të kualifikuar të metodës, duhet ta specifikoni një argument për parametrin e parë.
- Nëse tipi që po zgjerohet e përcakton një metodë instancë me emrin e njëjtë si metoda juaj zgjeruese dhe një nënshkrim (signature) kompatibil (të përputhshëm), metoda instancë e hijezon (shadows) metodën zgjeruese.

```

// Fig. 10.16: TestIZgjerimeveTeKohes.cs
// Demonstrimi i metodave zgjeruese.
using System;

class TestIZgjerimeveTeKohes
{
    static void Main()
    {
        var kohaIme = new Kohe2(); // thirre konstruktorin Kohe2
        kohaIme.CaktojeKohen(11, 34, 15); // caktoje kohën në 11:34:15

        // testoje metodën zgjeruese ShfaqjeKohen
        Console.WriteLine("Përdore metodën zgjeruese ShfaqjeKohen: ");
        kohaIme.ShfaqjeKohen();

        // testoje metodën zgjeruese ShtojiOret
        Console.WriteLine("Shtoji 5 orë me metodën zgjeruese ShtojiOret: ");
        var kohaEShtuar = kohaIme.ShtojiOret(5); // shtoji pesë orë
        kohaEShtuar.ShfaqjeKohen(); // shfaqje objektin e ri Kohe2

        // shtoji orët dhe shfaqje kohën në një urdhër
        Console.WriteLine("Shtoji 15 orë me metodën zgjeruese: ");
        kohaIme.ShtojiOret(15).ShfaqjeKohen(); // shtoji orët dhe shfaqje kohën

        // use fully qualified extension-method name to display the time
        Console.WriteLine("Përdore emrin emrin e kualifikuar plotësisht të metodës zgjeruese: ");
        ZgjerimetEKohes.ShfaqjeKohen(kohaIme);
        Console.ReadKey();
    }
}

// klasa e metodave zgjeruese
static class ZgjerimetEKohes
{
    // shfaqje objektin Kohe2 në konsolë (dritare komanduese)
    public static void ShfaqjeKohen(this Kohe2 njeKohe)
    {
        Console.WriteLine(njeKohe.ToString());
    }

    // shtoja kohës numrin e specifikuar të orëve
    // dhe ktheje një objekt të ri Kohe2
    public static Kohe2 ShtojiOret(this Kohe2 njeKohe, int oret)
    {
        // krijoje një objekt të ri Kohë2
        var kohaERe = new Kohe2()
        {
            Minuta = njeKohe.Minuta,
            Sekonda = njeKohe.Sekonda
        };

        // shtoja kohës së dhënë numrin e specifikuar të orëve
        kohaERe.Ora = (njeKohe.Ora + oret) % 24;

        return kohaERe; // ktheje objektin e ri Kohe2
    }
}

```

11. Programimi i Orientuar në Objekte: Trashëgimia

11.1 Hyrje

- Trashëgimia është formë e ripërdorimit të softuerit në të cilën një klasë e re krijohet duke i përvetësuar (absorbuar) anëtarët e një klase ekzistuese dhe duke i avancuar ata me aftësi të reja apo të modifikuara. Me trashëgimi, kurseni kohë gjatë zhvillimit të aplikacionit duke ripërdorur softuer me cilësi të lartë që është i dëshmuar dhe i pastruar nga defektet (bug-at).
- Një klasë e derivuar (e prejardhur, e nxjerrë) është më specifike se klasa e saj bazë dhe e përfaqëson një grup më të specializuar të objekteve.
- Relacioni (marrëdhënia) **is-a (është [një])** përfaqëson trashëgimi. Në relacion **is-a (është [një])** një objekt i një klase të derivuar mund të trajtohet edhe si objekt i klasës së tij bazë.

11.2 Klasat bazë dhe klasat e derivuara (të prejardhura, të nxjerra)

- Relacionet e trashëgimisë formojnë struktura hierarkike si pemë. Një klasë bazë ekziston në relacion hierarkik me klasat e saj të derivuara.
- Objektet e të gjitha klasave që e zgjerojnë një klasë të përbashkët bazë mund të trajtohen si objekte të asaj klase bazë. Mirëpo, objektet e klasës bazë nuk mund të trajtohen si objekte të klasave të tyre të derivuara.
- Kur një metodë e klasës bazë trashëgohet nga një klasë e derivuar, ajo klasë e derivuar shpesh ka nevojë për një version të përshtatur (customized) të metodës. Në raste të tilla, klasa e derivuar mund ta mbishkruajë metodën e klasës bazë me implementim të duhur.

Klasa bazë	Klasat e derivuara
Student	StudentIBachelorit, StudentIMasterit
Formë	Rreth, Trekëndësh, Drejtkëndësh
Kredi	KrediPërVeturë, KrediPërRiparimTëShtëpisë, KrediMeHipotekë
Punonjës	Mësimdhënës, NëpunësNëKompani, PunëtorMeOrë, PunëtorMeShtesaNëPagë
ObjektiHapësirës	Yll, Hënë, Planet, PjatëFluturuese
LlogariEBankës	LlogariQarkulluese, LlogariEKursimeve

Fig. 11.1 | Shembuj të trashëgimisë.

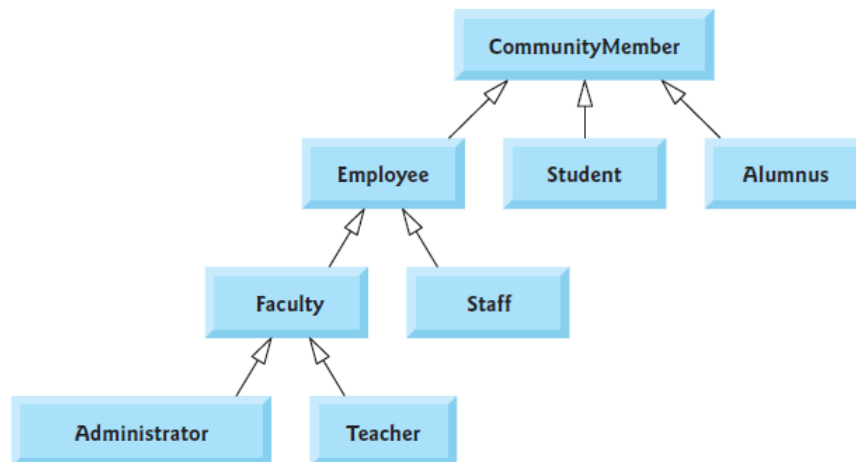


Fig. 11.2 | Inheritance hierarchy UML class diagram for university CommunityMembers.

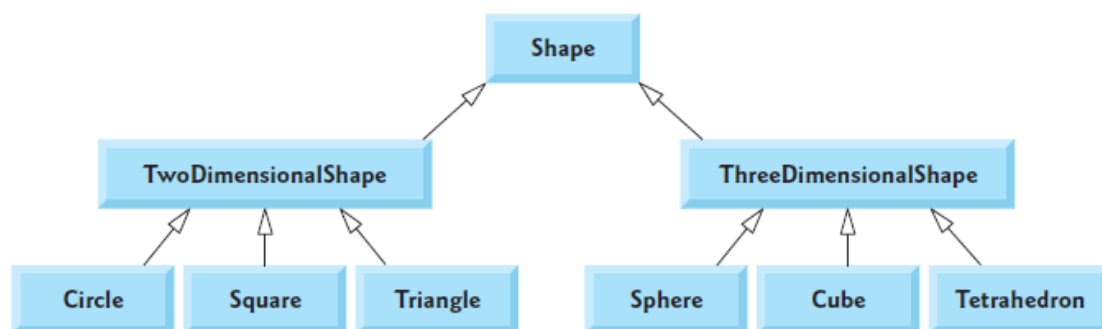


Fig. 11.3 | UML class diagram showing an inheritance hierarchy for Shapes.

11.3 Anëtarët e mbrojtur (*protected*)

- Përdorimi i qasjes **protected** (e mbrojtur) e ofron një nivel të ndërmjetëm të qasjes ndërmjet **public** (publike) dhe **private**. Anëtarët **protected** (të mbrojtur) të një klase bazë mund të qasen nga anëtarët e asaj klase bazë **dhe** nga anëtarët e klasave të saj të prejardhura.
- Metodat e një klase të derivuar (të prejardhur) nuk mund t'u qasen drejtpërdrejt anëtarëve **private** (privatë) të klasës bazë.

11.4. Relacioni ndërmjet klasave bazë dhe klasave të derivuara

11.4.1 Krijimi dhe përdorimi i një klase

```
// Fig. 11.4: PunonjesMeKomision.cs
// Klasa PunonjesMeKomision e përfaqëson një punonjës me komision.
using System;

public class PunonjesMeKomision : object
{
    private string Emri { get; }
    private string Mbiemri { get; }
    private string NumriPersonal { get; }
    private decimal shitjetBruto; // shitjet bruto javore
    private decimal normaEKomisionit; // përqindja e komisionit

    // konstruktor me pesë parametra
    public PunonjesMeKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEKomisionit)
    {
        // thirrja implicite e konstruktorit të klasës object ndodh këtu
        Emri = emri;
        Mbiemri = mbiemri;
        NumriPersonal = numriPersonal;
        ShitjetBruto = shitjetBruto; // i validon shitjet bruto
        NormaEKomisionit = normaEKomisionit; // e validon normën e komisionit
    }

    // properti që i merr (gets) dhe i cakton (sets)
    // shitjet bruto të punonjësit
    public decimal ShitjetBruto
    {
        get
        {
            return shitjetBruto;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(ShitjetBruto)} duhet të jetë >= 0");
            }

            shitjetBruto = value;
        }
    }
}
```

```
// properti që e merr (gets) dhe e cakton (sets)
// normën e komisionit të punonjësit
public decimal NormaEKomisionit
{
    get
    {
        return normaEKomisionit;
    }
    set
    {
        if (value <= 0 || value >= 1) // validimi
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(NormaEKomisionit)} duhet të jetë > 0 dhe < 1");
        }

        normaEKomisionit = value;
    }
}

// llogarite pagën e punonjësit me komision
public decimal Fitimet() => normaEKomisionit * shitjetBruto;

// ktheje paraqitjen string të objektit PunonjesMeKomision
public override string ToString() =>
    $"punonjësi me komision: {Emri} {Mbiemri}\n" +
    $"numri personal: {NumriPersonal}\n" +
    $"paga bruto: {shitjetBruto:C}\n" +
    $"norma e komisionit: {normaEKomisionit:F2}";
}
```

```
// Fig. 11.5: TestIPunonjesMeKomision.cs
// Testimi i klasës PunonjesMeKomision.
using System;

class TestIPunonjesMeKomision
{
    static void Main()
    {
        // instancinoje objektin PunonjesMeKomision
        var punonjesi = new PunonjesMeKomision("Sue", "Jones",
            "222-22-2222", 10000.00M, .06M);

        // shfaq shënimet e PunonjesitMeKomision
        Console.WriteLine(
            "Informacioni i punonjësit i marrë nga vetitë dhe metodat: \n");
        Console.WriteLine($"Emri është {punonjesi.Emri}");
        Console.WriteLine($"Mbiemri është {punonjesi.Mbiemri}");
        Console.WriteLine(
            $"Numri personal është {punonjesi.NumriPersonal}");
        Console.WriteLine($"Shitjet bruto janë {punonjesi.ShitjetBruto:C}");
        Console.WriteLine(
            $"Norma e komisionit është {punonjesi.NormaEKomisionit:F2}");
        Console.WriteLine($"Fitimet janë {punonjesi.Fitimet():C}");

        punonjesi.ShitjetBruto = 5000.00M; // caktoji shitjet bruto
        punonjesi.NormaEKomisionit = .1M; // caktoje normën e komisionit

        Console.WriteLine(
            "\nInformacioni i përditësuar i marrë nga ToString:\n");
        Console.WriteLine(punonjesi);
        Console.WriteLine($"fitimet: {punonjesi.Fitimet():C}");
        Console.ReadKey();
    }
}
```

```
Informacioni i punonjësit i marrë nga vetitë dhe metodat:

Emri është Sue
Mbiemri është Jones
Numri personal është 222-22-2222
Shitjet bruto janë £10,000.00
Norma e komisionit është 0.06
Fitimet janë £600.00

Informacioni i përditësuar i marrë nga ToString:

punonjësi me komision: Sue Jones
numri personal: 222-22-2222
shitjet bruto: £5,000.00
norma e komisionit: 0.10
fitimet: £500.00
```

- Një dypikësh (:) i pasuar nga një emër i klasës bazë në fund të ballinës (header-it) të deklarimit të klasës tregon se klasa e deklaruar e zgjeron klasën bazë.
- Nëse një klasë nuk specifikon se trashëgon nga ndonjë klasë tjetër, klasa në mënyrë implicite trashëgon nga **object**.
- Puna (detyra) e parë e çdo konstruktori të klasës së derivuar është ta thërrasë konstruktorin e klasës bazë, ose në mënyrë eksplicite ose në mënyrë implicite (nëse nuk është specifikuar asnjë thirrje e konstruktorit).
- Konstruktorët nuk trashëgohen. Edhe nëse një klasë nuk ka konstruktorë, konstruktori i paracaktuar (default) të cilin kompajleri e deklaron në mënyrë implicite për klasën do ta thërrasë konstruktorin e paracaktuar të klasës bazë apo konstruktorin pa parametra.
- Metoda **ToString** është një nga metodat që secila klasë e trashëgon direkt (drejtpërdrejt) ose indirekt (tërthorazi) nga klasa **object**, e cila është rrënja e hierakisë së klasave të C#.
- Për ta mbishkruar (override) një metodë të klasës bazë, një klasë e derivuar duhet ta deklarojnë një metodë me fjalën kyçe **override** dhe me nënshkrimin (signature) e njëjtë (emri i metodës, numri i parametarve dhe tipet e parametrave) dhe tipin kthyes sikur metoda e klasës bazë. Po ashtu, metoda e klasës bazë duhet të deklarohet **virtual**.
- Është gabim i kompajlimit të mbishkruhet një metodë me modifikues të ndryshëm të qasjes (aksesit).

11.4.2 Krijimi i një klase pa e përdorur trashëgiminë

```
// Fig. 11.6: PunonjesMeBazePlusKomision.cs
// Klasa PunonjesMeBazePlusKomision e përfaqëson një punonjës
// që e pranon një pagë bazë plus komisionin
using System;

public class PunonjesMeBazePlusKomision
{
    public string Emri { get; }
    public string Mbiemri { get; }
    public string NumriPersonal { get; }
    private decimal shitjetBruto; // shitjet javore bruto
    private decimal normaEKomisionit; // përqindja e komisionit
    private decimal pagaBaze; // paga bazë për javë

    // konstruktor me gjashtë parametra
    public PunonjesMeBazePlusKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEKomisionit, decimal pagaBaze)
    {
        // thirrja implicite e konstruktorit të klasës object ndodh këtu
        Emri = emri;
        Mbiemri = mbiemri;
        NumriPersonal = numriPersonal;
        ShitjetBruto = shitjetBruto; // i validon shitjet bruto
        NormaEKomisionit = normaEKomisionit; // e validon normën e komisionit
        PagaBaze = pagaBaze; // e validon pagën bazë
    }

    // properti që i merr (gets) dhe i cakton (sets) shitjet bruto
    public decimal ShitjetBruto
    {
        get
        {
            return shitjetBruto;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(ShitjetBruto)} duhet të jetë >= 0");
            }

            shitjetBruto = value;
        }
    }
}
```

```

// properti që e merr (gets) dhe e cakton (sets)
// normën e shtesës (komisionit)
public decimal NormaEKomisionit
{
    get
    {
        return normaEKomisionit;
    }
    set
    {
        if (value <= 0 || value >= 1) // validation
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(NormaEKomisionit)} duhet të jetë > 0 dhe < 1");
        }

        normaEKomisionit = value;
    }
}

// properti që e merr (gets) dhe e cakton (sets)
// pagën bazë të PunonjesitMeBazePlusShtesaNePage
public decimal PagaBaze
{
    get
    {
        return pagaBaze;
    }
    set
    {
        if (value < 0) // validimi
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(PagaBaze)} duhet të jetë >= 0");
        }

        pagaBaze = value;
    }
}

// llogarit fitimet
public decimal Fitimet() =>
    pagaBaze + (normaEKomisionit * shitjetBruto);

// ktheje paraqitjen string të objektit PunonjesMeBazePlusKomision
public override string ToString() =>
    $"punonjësi me bazë plus shtesa në pagë: {Emri} {Mbiemri}\n" +
    $"numri personal: {NumriPersonal}\n" +
    $"shitjet bruto: {shitjetBruto:C}\n" +
    $"norma e komisionit: {normaEKomisionit:F2}\n" +
    $"paga bazë: {pagaBaze:C}";
}

```

```
// Fig. 11.7: TestIPunonjesMeBazePlusKomision.cs
// Testimi i klasës PunonjesMeBazePlusKomision
using System;

class TestIPunonjesMeBazePlusKomision
{
    static void Main()
    {
        // instancinoje objektin PunonjesMeBazePlusKomision
        var punonjesi = new PunonjesMeBazePlusKomision("Bob", "Lewis",
            "333-33-3333", 5000.00M, .04M, 300.00M);

        // shfaq shënimet e objektit PunonjesMeBazePlusKomision
        Console.WriteLine(
            "Informacioni i punonjësit i marrë nga vetitë dhe metodat: \n");
        Console.WriteLine($"Emri është {punonjesi.Emri}");
        Console.WriteLine($"Mbiemri është {punonjesi.Mbiemri}");
        Console.WriteLine(
            $"Numri personal është {punonjesi.NumriPersonal}");
        Console.WriteLine($"Shitjet bruto janë {punonjesi.ShitjetBruto:C}");
        Console.WriteLine(
            $"Norma e komisionit është {punonjesi.NormaEKomisionit:F2}");
        Console.WriteLine($"Paga bazë është {punonjesi.PagaBaze:C}");
        Console.WriteLine($"Fitimet janë {punonjesi.Fitimet():C}");

        punonjesi.PagaBaze = 1000.00M; // caktoje pagën bazë

        Console.WriteLine(
            "\nInformacioni i përditësuar i marrë nga ToString:\n");
        Console.WriteLine(punonjesi);
        Console.WriteLine($"fitimet: {punonjesi.Fitimet():C}");
        Console.ReadKey();
    }
}
```

```
Informacioni i punonjësit i marrë nga vetitë dhe metodat:

Emri është Bob
Mbiemri është Lewis
Numri personal është 333-33-3333
Shitjet bruto janë £5,000.00
Norma e komisionit është 0.04
Paga bazë është £300.00
Fitimet janë £500.00

Informacioni i përditësuar i marrë nga ToString:

punonjësi me bazë plus shtesa në pagë: Bob Lewis
numri personal: 333-33-3333
shitjet bruto: £5,000.00
norma e komisionit: 0.04
paga bazë: £1,000.00
fitimet: £1,200.00
```

- Kopjimi dhe pastimi (“ngjitja me ngjitës – paste”) i kodit nga një klasë te një tjetër mund të përhapë gabime nëpër shumë fajlla të kodit burimor. Për ta shmangur duplikimin e kodit (dhe me gjasë edhe gabimet) në situatat kur dëshironi që një klasë t’i “përvetësojë” anëtarët e një klase tjetër, përdoreni trashëgiminë.

11.4.3 Krijimi i një hierarkie të trashëgimisë

- Fjalët kyçe **virtual** dhe **abstract** tregojnë se një properti (veti) e klasës bazë mund të mbishkruhet në klasë të derivuara.
- Modifikuesi (ndryshuesi) **override** deklaron se një metodë e klasës së derivuar e mbishkruan një metodë **virtual**-e apo **abstract**-e të klasës bazë. Ky modifikues po ashtu e deklaron në mënyrë implicite metodën e klasës së derivuar si **virtual**-e.
- Kur anëtarët e një klase bazë janë **private**, anëtarët e një klase të derivuar nuk lejohet t’u qasen atyre.

```
// Fig. 11.4: PunonjesMeKomision.cs
// Klasa PunonjesMeKomision e përfaqëson një punonjës me komision.
using System;

public class PunonjesMeKomision : object
{
    private string Emri { get; }
    private string Mbiemri { get; }
    private string NumriPersonal { get; }
    private decimal shitjetBruto; // shitjet bruto javore
    private decimal normaEKomisionit; // përqindja e komisionit

    // konstruktor me pesë parametra
    public PunonjesMeKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEKomisionit)
    {
        // thirrja implicite e konstruktorit të klasës object ndodh këtu
        Emri = emri;
        Mbiemri = mbiemri;
        NumriPersonal = numriPersonal;
        ShitjetBruto = shitjetBruto; // i validon shitjet bruto
        NormaEKomisionit = normaEKomisionit; // e validon normën e komisionit
    }
}
```



```
// properti që i merr (gets) dhe i cakton (sets)
// shitjet bruto të punonjësit
public decimal ShitjetBruto
{
    get
    {
        return shitjetBruto;
    }
    set
    {
        if (value < 0) // validimi
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(ShitjetBruto)} duhet të jetë >= 0");
        }

        shitjetBruto = value;
    }
}

// properti që e merr (gets) dhe e cakton (sets)
// normën e komisionit të punonjësit
public decimal NormaEKomisionit
{
    get
    {
        return normaEKomisionit;
    }
    set
    {
        if (value <= 0 || value >= 1) // validimi
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(NormaEKomisionit)} duhet të jetë > 0 dhe < 1");
        }

        normaEKomisionit = value;
    }
}

// llogaritë pagën e punonjësit me komision
public decimal Fitimet() => normaEKomisionit * shitjetBruto;

// ktheje paraqitjen string të objektit PunonjesMeKomision
public override string ToString() =>
    $"punonjësi me komision: {Emri} {Mbiemri}\n" +
    $"numri personal: {NumriPersonal}\n" +
    $"paga bruto: {shitjetBruto:C}\n" +
    $"norma e komisionit: {normaEKomisionit:F2}";
}
```

```
// Fig. 11.5: TestIPunonjesMeKomision.cs
// Testimi i klasës PunonjesMeKomision.
using System;

class TestIPunonjesMeKomision
{
    static void Main()
    {
        // instancinoje objektin PunonjesMeKomision
        var punonjesi = new PunonjesMeKomision("Sue", "Jones",
            "222-22-2222", 10000.00M, .06M);

        // shfaq shënimet e PunonjesitMeKomision
        Console.WriteLine(
            "Informacioni i punonjësit i marrë nga vetitë dhe metodat: \n");
        Console.WriteLine($"Emri është {punonjesi.Emri}");
        Console.WriteLine($"Mbiemri është {punonjesi.Mbiemri}");
        Console.WriteLine(
            $"Numri personal është {punonjesi.NumriPersonal}");
        Console.WriteLine($"Shitjet bruto janë {punonjesi.ShitjetBruto:C}");
        Console.WriteLine(
            $"Norma e komisionit është {punonjesi.NormaEKomisionit:F2}");
        Console.WriteLine($"Fitimet janë {punonjesi.Fitimet():C}");

        punonjesi.ShitjetBruto = 5000.00M; // caktoji shitjet bruto
        punonjesi.NormaEKomisionit = .1M; // caktoje normën e komisionit

        Console.WriteLine(
            "\nInformacioni i përditësuar i marrë nga ToString:\n");
        Console.WriteLine(punonjesi);
        Console.WriteLine($"fitimet: {punonjesi.Fitimet():C}");
        Console.ReadKey();
    }
}
```

```
// Fig. 11.8: PunonjesMeBazePlusKomision.cs
// PunonjesMeBazePlusKomision trashëgon nga PunonjesMeKomision.
using System;

public class PunonjesMeBazePlusKomision : PunonjesMeKomision
{
    private decimal pagaBaze; // paga bazë për javë

    // konstuktori me gjashtë parametra i klasës së derivuar
    // me thirrje në konstruktorin e klasës bazë PunonjesMeKomision
    public PunonjesMeBazePlusKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEKomisionit, decimal pagaBaze)
        : base(emri, mbiemri, numriPersonal,
            shitjetBruto, normaEKomisionit)
    {
        PagaBaze = pagaBaze; // e validon pagën bazë
    }

    // properti që e merr (gets) dhe e cakton (sets)
    // pagën bazë të PunonjesitMeBazePlusShtesaNePage
    public decimal PagaBaze
    {
        get
        {
            return pagaBaze;
        }
        set
        {
            if (value < 0) // validation
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(PagaBaze)} duhet të jetë >= 0");
            }

            pagaBaze = value;
        }
    }

    // llogarit fitimet
    public override decimal Fitimet() =>
        pagaBaze + (normaEKomisionit * shitjetBruto);

    // ktheje paraqitjen string të objektit PunonjesMeBazePlusKomision
    public override string ToString() =>
        // nuk lejohet: tenton t'u qaset anëtarëve privatë të klasës bazë
        $"punonjësi me bazë plus komision: {Emri} {Mbiemri}\n" +
        $"numri personal: {NumriPersonal}\n" +
        $"shitjet bruto: {shitjetBruto:C}\n" +
        $"norma e komisionit: {normaEKomisionit:F2}\n" +
        $"paga bazë: {pagaBaze:C}";
}
}
```

Error List					
Entire Solution		8 Errors	0 Warnings	0 Messages	Build + IntelliSense
	Code	Description	Project	File	Line
	CS0506	'PunonjesMeBazePlusKomision.Fitimet()': cannot override inherited member 'PunonjesMeKomision.Fitimet()' because it is not marked virtual, abstract, or override	BasePlusCommissionEmpl...	PunonjesMeBazePlusKomi...	41
	CS0122	'PunonjesMeKomision.normaEKomisionit' is inaccessible due to its protection level	BasePlusCommissionEmpl...	PunonjesMeBazePlusKomi...	42
	CS0122	'PunonjesMeKomision.shitjetBruto' is inaccessible due to its protection level	BasePlusCommissionEmpl...	PunonjesMeBazePlusKomi...	42
	CS0122	'PunonjesMeKomision.Emri' is inaccessible due to its protection level	BasePlusCommissionEmpl...	PunonjesMeBazePlusKomi...	47
	CS0122	'PunonjesMeKomision.Mbiemri' is inaccessible due to its protection level	BasePlusCommissionEmpl...	PunonjesMeBazePlusKomi...	47
	CS0122	'PunonjesMeKomision.NumriPersonal' is inaccessible due to its protection level	BasePlusCommissionEmpl...	PunonjesMeBazePlusKomi...	48
	CS0122	'PunonjesMeKomision.shitjetBruto' is inaccessible due to its protection level	BasePlusCommissionEmpl...	PunonjesMeBazePlusKomi...	49
	CS0122	'PunonjesMeKomision.normaEKomisionit' is inaccessible due to its protection level	BasePlusCommissionEmpl...	PunonjesMeBazePlusKomi...	50

11.4.4 Hierarkia e trashëgimisë duke përdorur variabla të instancës që janë protected

- Trashëgimi i variablave të instancës që janë **protected** (të mbrojtura) jua mundëson t'u qaseni variablave në klasën e derivuar pa i thirrur aksesorët (qasësit) **set** apo **get** të properti-t përkatës.
- Softueri thuhet se është i brishtë apo i thyeshëm kur një ndryshim i vogël në klasën bazë mund ta “prishë” apo “thejë” implementimin e klasës së derivuar. Duhet të jeni në gjendje ta ndryshoni implementimin e klasës bazë gjersa ende ua ofroni shërbimet e njëjta klasave të derivuara.
- Deklarimi i variablave të instancës të klasës bazë si **private** e mundëson që implementimi i klasës bazë i këtyre variablave të instancës të ndryshojë pa ndikuar në (pa i afektuar) implementimet e klasave të derivuara.

```
// Fig. 11.4: PunonjesMeKomision.cs
// Klasa PunonjesMeKomision e përfaqëson një punonjës me komision.
using System;

public class PunonjesMeKomision : object
{
    public string Emri { get; }
    public string Mbiemri { get; }
    public string NumriPersonal { get; }
    protected decimal shitjetBruto; // shitjet bruto javore
    protected decimal normaEKomisionit; // përqindja e komisionit

    // konstruktor me pesë parametra
    public PunonjesMeKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEKomisionit)
    {
        // thirrja implicite e konstruktorit të klasës object ndodh këtu
        Emri = emri;
        Mbiemri = mbiemri;
        NumriPersonal = numriPersonal;
        ShitjetBruto = shitjetBruto; // i validon shitjet bruto
        NormaEKomisionit = normaEKomisionit; // e validon normën e komisionit
    }

    // properti që i merr (gets) dhe i cakton (sets)
    // shitjet bruto të punonjësit
    public decimal ShitjetBruto
    {
        get
        {
            return shitjetBruto;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(ShitjetBruto)} duhet të jetë >= 0");
            }

            shitjetBruto = value;
        }
    }
}
```

```
// properti që e merr (gets) dhe e cakton (sets)
// normën e komisionit të punonjësit
public decimal NormaEKomisionit
{
    get
    {
        return normaEKomisionit;
    }
    set
    {
        if (value <= 0 || value >= 1) // validimi
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(NormaEKomisionit)} duhet të jetë > 0 dhe < 1");
        }

        normaEKomisionit = value;
    }
}

// llogarite pagën e punonjësit me komision
public virtual decimal Fitimet() => normaEKomisionit * shitjetBruto;

// ktheje paraqitjen string të objektit PunonjesMeKomision
public override string ToString() =>
    $"punonjësi me komision: {Emri} {Mbiemri}\n" +
    $"numri personal: {NumriPersonal}\n" +
    $"paga bruto: {shitjetBruto:C}\n" +
    $"norma e komisionit: {normaEKomisionit:F2}";
}
```

```
// Fig. 11.9: PunonjesMeBazePlusKomision.cs
// PunonjesMeBazePlusKomision trashëgon nga PunonjesMeKomision
// dhe ka qasje në anëtarët e mbrojtur (protected) të klasës PunonjesMeKomision
```

```
using System;
```

```
public class PunonjesMeBazePlusKomision : PunonjesMeKomision
{
    private decimal pagaBaze; // paga bazë për javë

    // konstruktor me gjashtë parametra i klasës së derivuar
    // me thirrje në konstruktorin e klasës bazë PunonjesMeShtesaNePage
    public PunonjesMeBazePlusKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEKomisionit, decimal pagaBaze)
        : base(emri, mbiemri, numriPersonal,
            shitjetBruto, normaEKomisionit)
    {
        PagaBaze = pagaBaze; // e validon pagën bazë
    }

    // properti që e merr (gets) dhe e cakton (sets)
    // pagën bazë të PunonjesitMeBazePlusShtesaNePage
    public decimal PagaBaze
    {
        get
        {
            return pagaBaze;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(PagaBaze)} duhet të jetë >= 0");
            }

            pagaBaze = value;
        }
    }

    // llogarit fitimet
    public override decimal Fitimet() =>
        pagaBaze + (normaEKomisionit * shitjetBruto);

    // return string representation of BasePlusCommissionEmployee
    public override string ToString() =>
        $"punonjësi me bazë plus shtesa në pagë: {Emri} {Mbiemri}\n" +
        $"numri personal: {NumriPersonal}\n" +
        $"shitjet bruto: {shitjetBruto:C}\n" +
        $"norma e komisionit: {normaEKomisionit:F2}\n" +
        $"paga bazë: {pagaBaze:C}";
}
}
```

```
// Fig. 11.7: TestIPunonjesMeBazePlusShtesaNePage.cs
// Testimi i klasës PunonjesMeBazePlusKomision
using System;

class TestIPunonjesMeBazePlusShtesaNePage
{
    static void Main()
    {
        // instancinoje objektin PunonjesMeBazePlusKomision
        var punonjesi = new PunonjesMeBazePlusKomision("Bob", "Lewis",
            "333-33-3333", 5000.00M, .04M, 300.00M);

        // shfaq shënimet e objektit PunonjesMeBazePlusKomision
        Console.WriteLine(
            "Informacioni i punonjësit i marrë nga vetitë dhe metodat: \n");
        Console.WriteLine($"Emri është {punonjesi.Emri}");
        Console.WriteLine($"Mbiemri është {punonjesi.Mbiemri}");
        Console.WriteLine(
            $"Numri personal është {punonjesi.NumriPersonal}");
        Console.WriteLine($"Shitjet bruto janë {punonjesi.ShitjetBruto:C}");
        Console.WriteLine(
            $"Norma e komisionit është {punonjesi.NormaEKomisionit:F2}");
        Console.WriteLine($"Paga bazë është {punonjesi.PagaBaze:C}");
        Console.WriteLine($"Fitimet janë {punonjesi.Fitimet():C}");

        punonjesi.PagaBaze = 1000.00M; // caktoje pagën bazë

        Console.WriteLine(
            "\nInformacioni i përfitësuar i marrë nga ToString:\n");
        Console.WriteLine(punonjesi);
        Console.WriteLine($"fitimet: {punonjesi.Fitimet():C}");
        Console.ReadKey();
    }
}
```

```
Informacioni i punonjësit i marrë nga vetitë dhe metodat:

Emri është Bob
Mbiemri është Lewis
Numri personal është 333-33-3333
Shitjet bruto janë £5,000.00
Norma e komisionit është 0.04
Paga bazë është £300.00
Fitimet janë £500.00

Informacioni i përfitësuar i marrë nga ToString:

punonjësi me bazë plus shtesa në pagë: Bob Lewis
numri personal: 333-33-3333
shitjet bruto: £5,000.00
norma e komisionit: 0.04
paga bazë: £1,000.00
fitimet: £1,200.00
```


11.4.5 Hierarkia e trashëgimisë duke përdorur variabla të instancës që janë private

- Vendoseni fjalën kyçe **base** dhe operatorin e qasjes së anëtarit (.) para emrit të metodës së klasës bazë për ta thirrur një metodë të mbishkruar të klasës bazë nga klasa e derivuar.
- Dështimi për ta shkruar para emrit të metodës së klasës bazë – fjalën kyçe **base** dhe operatorin e qasjes së anëtarit (.) kur i referoheni metodës së klasës bazë shkakton që metoda e klasës së derivuar ta thërrasë veten, duke krijuar rekursion të pafund.

```
// Fig. 11.10: PunonjesMeKomision.cs
// Klasa PunonjesMeKomision e përfaqëson një punonjës me komision.
using System;

public class PunonjesMeKomision
{
    public string Emri { get; }
    public string Mbiemri { get; }
    public string NumriPersonal { get; }
    private decimal shitjetBruto; // shitjet bruto javore
    private decimal normaEKomisionit; // përqindja e komisionit

    // konstruktor me pesë parametra
    public PunonjesMeKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEKomisionit)
    {
        // thirrja implicite e konstruktorit të klasës object ndodh këtu
        Emri = emri;
        Mbiemri = mbiemri;
        NumriPersonal = numriPersonal;
        ShitjetBruto = shitjetBruto; // i validon shitjet bruto
        NormaEKomisionit = normaEKomisionit; // e validon normën e komisionit
    }

    // properti që i merr (gets) dhe i cakton (sets)
    // shitjet bruto me shtesa (komision) të punonjësit
    public decimal ShitjetBruto
    {
        get
        {
            return shitjetBruto;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(ShitjetBruto)} duhet të jetë >= 0");
            }

            shitjetBruto = value;
        }
    }
}
```

```
// properti që e merr (gets) dhe e cakton (sets)
// normën e shtesës (komisionit) të punonjësit
public decimal NormaEKomisionit
{
    get
    {
        return normaEKomisionit;
    }
    set
    {
        if (value <= 0 || value >= 1) // validimi
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(NormaEKomisionit)} duhet të jetë > 0 dhe < 1");
        }

        normaEKomisionit = value;
    }
}

// llogarite pagën e punonjësit me komision
public virtual decimal Fitimet() => NormaEKomisionit * ShitjetBruto;

// ktheje paraqitjen string të objektit PunonjesMeKomision
public override string ToString() =>
    $"punonjësi me komision: {Emri} {Mbiemri}\n" +
    $"numri personal: {NumriPersonal}\n" +
    $"shitjet bruto: {ShitjetBruto:C}\n" +
    $"norma e komisionit: {NormaEKomisionit:F2}";
}
```

```
// Fig. 11.11: PunonjesMeBazePlusKomision.cs
// PunonjesMeBazePlusKomision trashëgon nga PunonjesMeKomision
// dhe ka qasje të kontrolluar në anëtarët privatë të klasës PunonjesMeKomision
// përmes propertive të saj publike
```

```
using System;
```

```
public class PunonjesMeBazePlusKomision : PunonjesMeKomision
{
    private decimal pagaBaze; // paga bazë për javë

    // konstruktor me gjashtë parametra i klasës së derivuar
    // me thirrje në konstruktorin e klasës bazë PunonjesMeKomision
    public PunonjesMeBazePlusKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEShteses, decimal pagaBaze)
        : base(emri, mbiemri, numriPersonal,
            shitjetBruto, normaEShteses)
    {
        PagaBaze = pagaBaze; // e validon pagën bazë
    }

    // properti që e merr (gets) dhe e cakton (sets)
    // pagën bazë të PunonjesitMeBazePlusShtesaNePage
    public decimal PagaBaze
    {
        get
        {
            return pagaBaze;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(PagaBaze)} duhet të jetë >= 0");
            }

            pagaBaze = value;
        }
    }

    // llogarit fitimet
    public override decimal Fitimet() => PagaBaze + base.Fitimet();

    // ktheje paraqitjen string të objektit PunonjesMeBazePlusKomision
    public override string ToString() =>
        $"me pagë bazë {base.ToString()}\\npaga bazë: {PagaBaze:C}";
}
```

```
// Fig. 11.7: TestIPunonjesMeBazePlusKomision.cs
// Testimi i klasës PunonjesMeBazePlusKomision
using System;

class TestIPunonjesMeBazePlusKomision
{
    static void Main()
    {
        // instancinoje objektin PunonjesMeBazePlusKomision
        var punonjesi = new PunonjesMeBazePlusKomision("Bob", "Lewis",
            "333-33-3333", 5000.00M, .04M, 300.00M);

        // shfaq shënimet e objektit PunonjesMeBazePlusKomision
        Console.WriteLine(
            "Informacioni i punonjësit i marrë nga vetitë dhe metodat: \n");
        Console.WriteLine($"Emri është {punonjesi.Emri}");
        Console.WriteLine($"Mbiemri është {punonjesi.Mbiemri}");
        Console.WriteLine(
            $"Numri personal është {punonjesi.NumriPersonal}");
        Console.WriteLine($"Shitjet bruto janë {punonjesi.ShitjetBruto:C}");
        Console.WriteLine(
            $"Norma e komisionit është {punonjesi.NormaEKomisionit:F2}");
        Console.WriteLine($"Paga bazë është {punonjesi.PagaBaze:C}");
        Console.WriteLine($"Fitimet janë {punonjesi.Fitimet():C}");

        punonjesi.PagaBaze = 1000.00M; // caktoje pagën bazë

        Console.WriteLine(
            "\nInformacioni i përditësuar i marrë nga ToString:\n");
        Console.WriteLine(punonjesi);
        Console.WriteLine($"fitimet: {punonjesi.Fitimet():C}");
        Console.ReadKey();
    }
}
```

```
Informacioni i punonjësit i marrë nga vetitë dhe metodat:

Emri është Bob
Mbiemri është Lewis
Numri personal është 333-33-3333
Shitjet bruto janë £5,000.00
Norma e komisionit është 0.04
Paga bazë është £300.00
Fitimet janë £500.00

Informacioni i përditësuar i marrë nga ToString:

me pagë bazë punonjësi me komision: Bob Lewis
numri personal: 333-33-3333
shitjet bruto: £5,000.00
norma e komisionit: 0.04
paga bazë: £1,000.00
fitimet: £1,200.00
```

11.5 Konstruktoret në klasat e derivuara

- Instancimi i një objekti të klasës së derivuar e nis një zinxhir të thirrjeve të konstruktorëve. Konstruktori i fundit i thirrur në zinxhir është gjithmonë konstruktori për klasën **object**. Trupi i konstruktorit të klasës fillestare (zanafillëse, origjinale) të derivuar e përfundon ekzekutimin i fundit.

11.6 Inxhinierimi i softuerit me trashëgimi

- Mund t'i përshtasim (customize) klasat e reja për t'i përmbushur nevojat tona duke përfshirë anëtarë shtesë dhe duke i mbishkruar anëtarët e klasës bazë.

11.7 Klasa object

- Të gjitha klasat në C# trashëgojnë direkt (drejtpërdrejt) apo indirekt (tërthorazi) nga klasa **object**, kështu që shtatë metodat e saj trashëgohen nga të gjitha klasat e tjera. Këto metoda janë: **Equals**, **Finalize**, **GetHashCode**, **GetType**, **MemberwiseClone**, **ReferenceEquals** dhe **ToString**.

Method	Description
Equals	This method compares the current object to another object for equality and returns <code>true</code> if they're equal and <code>false</code> otherwise. It takes any object as an argument. When objects of a particular class must be compared for equality, the class should override method <code>Equals</code> to compare the <i>contents</i> of the two objects. The website http://bit.ly/OverridingEqualsCSharp explains the requirements for a properly overridden <code>Equals</code> method.
Finalize	This method cannot be explicitly declared or called. When a class contains a destructor, the compiler implicitly renames it to override the protected method <code>Finalize</code> , which is called <i>only</i> by the garbage collector before it reclaims an object's memory. The garbage collector is not guaranteed to reclaim an object, thus it's <i>not</i> guaranteed that an object's <code>Finalize</code> method will execute. When a derived class's <code>Finalize</code> method executes, it performs its task, then invokes the base class's <code>Finalize</code> method. In general, you should avoid using <code>Finalize</code> .

Fig. 11.12 | **object** methods that are inherited directly or indirectly by all classes. (Part 1 of 2.)

Method	Description
<code>GetHashCode</code>	A hashtable data structure relates objects, called <i>keys</i> , to corresponding objects, called <i>values</i> . We discuss <code>Hashtable</code> in Chapter 21. When a value is initially inserted in a hashtable, the key's <code>GetHashCode</code> method is called. The value returned is used by the hashtable to determine the location at which to insert the corresponding value. The key's hashcode is also used by the hashtable to locate the key's corresponding value.
<code>GetType</code>	Every object knows its own type at execution time. Method <code>GetType</code> (used in Section 12.5) returns an object of class <code>Type</code> (namespace <code>System</code>) that contains information about the object's type, such as its class name (obtained from <code>Type</code> property <code>FullName</code>).
<code>MemberwiseClone</code>	This protected method, which takes no arguments and returns an object reference, makes a copy of the object on which it's called. The implementation of this method performs a <i>shallow copy</i> —instance-variable values in one object are copied into another object of the same type. For reference types, only the references are copied.
<code>ReferenceEquals</code>	This static method receives two object references and returns <code>true</code> if they're the same instance or if they're <code>null</code> references. Otherwise, it returns <code>false</code> .
<code>ToString</code>	This method (introduced in Section 7.4) returns a string representation of the current object. The default implementation of this method returns the namespace followed by a dot and the class name of the object's class.

Fig. 11.12 | object methods that are inherited directly or indirectly by all classes. (Part 2 of 2.)

12. Programimi i Orientuar në Objekte: Polimorfizmi dhe Interfejsat (Shumëformësia dhe Ndërfaqet)

12.1 Hyrje

- Me polimorfizëm (shumëformësi), mund të dizajnojmë (projektojmë) dhe të implementojmë sisteme që janë të zgjerueshme lehtësisht – klasat e reja mund të shtohen me pak apo aspak ndryshim (modifikim) në pjesët e përgjithshme të aplikacionit.

12.2 Shembuj të polimorfizmit

- Me polimorfizëm, emri dhe nënshkrimi i njëjtë i metodës mund të përdoret për të shkaktuar veprime të ndryshme, varësisht nga tipi i objektit në të cilin thirret metoda.
- Polimorfizmi e promovon zgjerueshmërinë: Softueri që e thërret sjelljen polimorfike është i pavarur nga tipet e objekteve të cilave ua dërgon mesazhet. Tipet e reja të objekteve që mund t'u përgjigjen thirrjeve ekzistuese të metodave mund të inkorporohen (përfshihen) në sistem pa kërkuar ndryshim (modifikim) të sistemit bazë.

12.3 Demonstrimi i sjelljes polimorfike

- Thirrja e një metode në një objekt të klasës së derivuar përmes një reference të klasës bazë e thërret funksionalitetin e klasës së derivuar – tipi i objektit të referencuar e përcakton se cila metodë thirret.

(Shih Fig. 12.1 më poshtë për shembullin e ri)

```
// Fig. 11.10: PunonjesMeKomision.cs
// Klasa PunonjesMeKomision e përfaqëson një punonjës me komision.
using System;

public class PunonjesMeKomision
{
    public string Emri { get; }
    public string Mbiemri { get; }
    public string NumriPersonal { get; }
    private decimal shitjetBruto; // shitjet bruto javore
    private decimal normaEKomisionit; // përqindja e komisionit

    // konstruktor me pesë parametra
    public PunonjesMeKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEKomisionit)
    {
        // thirrja implicite e konstruktorit të klasës object ndodh këtu
        Emri = emri;
        Mbiemri = mbiemri;
        NumriPersonal = numriPersonal;
        ShitjetBruto = shitjetBruto; // i validon shitjet bruto
        NormaEKomisionit = normaEKomisionit; // e validon normën e komisionit
    }

    // properti që i merr (gets) dhe i cakton (sets)
    // shitjet bruto me shtesa (komision) të punonjësit
    public decimal ShitjetBruto
    {
        get
        {
            return shitjetBruto;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(ShitjetBruto)} duhet të jetë >= 0");
            }

            shitjetBruto = value;
        }
    }
}
```



```
// properti që e merr (gets) dhe e cakton (sets)
// normën e shtesës (komisionit) të punonjësit
public decimal NormaEKomisionit
{
    get
    {
        return normaEKomisionit;
    }
    set
    {
        if (value <= 0 || value >= 1) // validimi
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(NormaEKomisionit)} duhet të jetë > 0 dhe < 1");
        }

        normaEKomisionit = value;
    }
}

// llogarite pagën e punonjësit me komision
public virtual decimal Fitimet() => NormaEKomisionit * ShitjetBruto;

// ktheje paraqitjen string të objektit PunonjesMeKomision
public override string ToString() =>
    $"punonjësi me komision: {Emri} {Mbiemri}\n" +
    $"numri personal: {NumriPersonal}\n" +
    $"shitjet bruto: {ShitjetBruto:C}\n" +
    $"norma e komisionit: {NormaEKomisionit:F2}";
}
```

```
// Fig. 11.11: PunonjesMeBazePlusKomision.cs
// PunonjesMeBazePlusKomision trashëgon nga PunonjesMeKomision
// dhe ka qasje të kontrolluar në anëtarët privatë të klasës PunonjesMeKomision
// përmes propertive të saj publike

using System;

public class PunonjesMeBazePlusKomision : PunonjesMeKomision
{
    private decimal pagaBaze; // paga bazë për javë

    // konstruktor me gjashtë parametra i klasës së derivuar
    // me thirrje në konstruktorin e klasës bazë PunonjesMeKomision
    public PunonjesMeBazePlusKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEShteses, decimal pagaBaze)
        : base(emri, mbiemri, numriPersonal,
            shitjetBruto, normaEShteses)
    {
        PagaBaze = pagaBaze; // e validon pagën bazë
    }

    // properti që e merr (gets) dhe e cakton (sets)
    // pagën bazë të PunonjesitMeBazePlusShtesaNePage
    public decimal PagaBaze
    {
        get
        {
            return pagaBaze;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(PagaBaze)} duhet të jetë >= 0");
            }

            pagaBaze = value;
        }
    }

    // llogarit fitimet
    public override decimal Fitimet() => PagaBaze + base.Fitimet();

    // ktheje paraqitjen string të objektit PunonjesMeBazePlusKomision
    public override string ToString() =>
        $"me pagë bazë {base.ToString()}\\npaga bazë: {PagaBaze:C}";
}
```

```

// Fig. 12.1: TestIPolimorfizmit.cs
// Caktimi i referencave të klasës bazë dhe të klasës së derivuar -
// variablave të klasës së derivuar.
using System;

class TestIPolimorfizmit
{
    static void Main()
    {
        // ndaja (caktoja) referencën e klasës bazë - variablës së klasës bazë
        var punonjesiMeKomision = new PunonjesMeKomision(
            "Sue", "Jones", "222-22-2222", 10000.00M, .06M);

        // ndaja (caktoja) referencën e klasës së derivuar - variablës së klasës së derivuar
        var punonjesiMeBazePlusKomision = new PunonjesMeBazePlusKomision(
            "Bob", "Lewis", "333-33-3333", 5000.00M, .04M, 300.00M);

        // thirre ToString dhe Fitimet në objektin e klasës bazë
        // duke e përdorur variablën e klasës bazë
        Console.WriteLine(
            "Thirri metodat ToString dhe Fitimet të klasës PunonjesMeShtesaNePage " +
            "me referencë të klasës bazë në objektin e klasës bazë\n");
        Console.WriteLine(punonjesiMeKomision.ToString());
        Console.WriteLine($"fitimet: {punonjesiMeKomision.Fitimet()}\n");

        // thirre ToString dhe Fitimet në objektin e klasës së derivuar
        // duke e përdorur variablën e klasës së derivuar
        Console.WriteLine("Thirri metodat ToString dhe Fitimet të klasës " +
            "PunonjesMeBazePlusKomision me referencë të klasës së derivuar " +
            "në objektin e klasës së derivuar\n");
        Console.WriteLine(punonjesiMeBazePlusKomision.ToString());
        Console.WriteLine(
            $"fitimet: {punonjesiMeBazePlusKomision.Fitimet()}\n");

        // thirre ToString dhe Fitimet në objektin e klasës së derivuar
        // duke e përdorur variablën e klasës bazë
        PunonjesMeKomision punonjesiMeKomision2 = punonjesiMeBazePlusKomision;
        Console.WriteLine(
            "Thirri metodat ToString dhe Fitimet të klasës PunonjesMeBazePlusKomision " +
            "me referencë të klasës bazë në objektin e klasës së derivuar\n");
        Console.WriteLine(punonjesiMeKomision2.ToString());
        Console.WriteLine(
            $"fitimet: {punonjesiMeBazePlusKomision.Fitimet()}\n");
        Console.ReadKey();
    }
}

```

```

Thirrri metodat ToString dhe Fitimet të klasës PunonjesMeShtesaNePage me
referencë të klasës bazë në objektin e klasës bazë

punonjësi me komision: Sue Jones
numri personal: 222-22-2222
shitjet bruto: £10,000.00
norma e komisionit: 0.06
fitimet: 600.0000

Thirrri metodat ToString dhe Fitimet të klasës PunonjesMeBazePlusKomision
me referencë të klasës së derivuar në objektin e klasës së derivuar

me pagë bazë punonjësi me komision: Bob Lewis
numri personal: 333-33-3333
shitjet bruto: £5,000.00
norma e komisionit: 0.04
paga bazë: £300.00
fitimet: 500.0000

Thirrri metodat ToString dhe Fitimet të klasës PunonjesMeBazePlusKomision
me referencë të klasës bazë në objektin e klasës së derivuar

me pagë bazë punonjësi me komision: Bob Lewis
numri personal: 333-33-3333
shitjet bruto: £5,000.00
norma e komisionit: 0.04
paga bazë: £300.00
fitimet: 500.0000

```

12.4 Klasat dhe metodat abstrakte

- Klasat abstrakte janë klasa të paplota për të cilat kurrë nuk synoni të instanconi objekte.
- Destinimi (qëllimi) i një klase abstrakte është në radhë të parë për ta ofruar një klasë bazë të përshtatshme nga e cila mund të trashëgojnë klasat e tjera, dhe kështu ta bashkë-përdorin (ta ndajnë) një dizajn të përbashkët.
- Klasat që mund të përdoren për të instancuar objekte quhen klasa konkrete.
- Një klasë e bëni abstrakte duke e deklaruar me fjalën kyçe **abstract**.
- Çdo klasë konkrete e derivuar e një klase bazë abstrakte duhet të ofrojë implementime konkrete të metodave dhe proprietive **abstract**-e të klasës bazë.
- Dështimi për t'i implementuar metodat dhe properti-t **abstract**-e të klasës bazë në një klasë të derivuar është gabim i kompajlimit pos nëse edhe klasa e derivuar deklarohet **abstract**-e.
- Edhe pse nuk mund të instancojmë objekte të klasave bazë **abstract**-e, mund t'i përdorim ato për të deklaruar variabla që mund të mbajnë referenca për objektet e cilësdo klasë konkrete të derivuar nga ato klasa **abstract**-e.

12.5 Studim i rastit: Sistemi i pagave duke përdorur polimorfizëm

E krijojmë një hierarki të avancuar të punonjësve për ta zgjidhur problemin vijues:

1. Punonjësit me pagë paguhen me pagë javore fikse pavarësisht nga numri i orëve të punuara.
2. Punonjësit me orë paguhen sipas orëve dhe marrin pagesë “një-herë-e-gjysmë” jashtë orarit për të gjitha orët e punuara mbi 40 orë.
3. Punonjësit me komision (shtesa) paguhen me përqindje të shitjeve të tyre.
4. Punonjësit me pagë dhe komision e marrin një pagë bazë plus një përqindje të shitjeve të tyre.

Për periudhën aktuale të pagave, kompania ka vendosur t’i shpërblejë punonjësit me pagë dhe komision duke ua shtuar 10% pagave të tyre bazë. Kompania dëshiron ta implementojë një aplikacion që i kryen llogaritjet e tij të pagave polimorfikisht (në mënyrë polimorfike, shumë-formëshe).

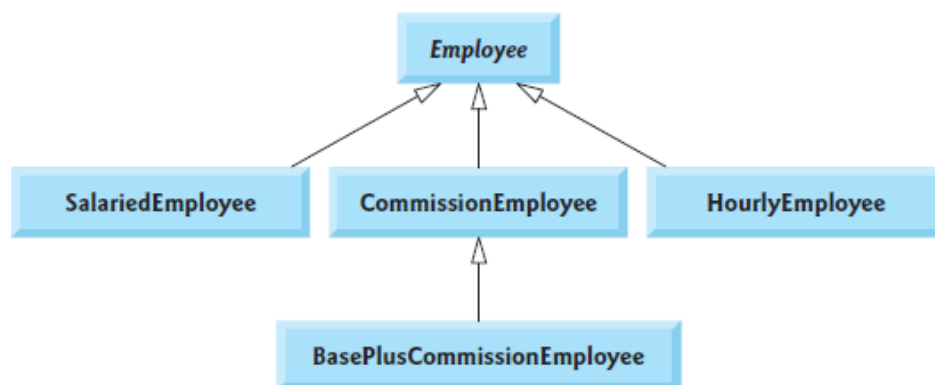


Fig. 12.2 | Employee hierarchy UML class diagram.

	Earnings	ToString
Employee	abstract	<i>firstName lastName</i> social security number: <i>SSN</i>
Salaried- Employee	weeklySalary	salaried employee: <i>firstName lastName</i> social security number: <i>SSN</i> weekly salary: <i>weeklysalar</i>
Hourly- Employee	<i>If hours <= 40</i> <i>wage * hours</i> <i>If hours > 40</i> <i>40 * wage +</i> <i>(hours - 40) *</i> <i>wage * 1.5</i>	hourly employee: <i>firstName lastName</i> social security number: <i>SSN</i> hourly wage: <i>wage</i> hours worked: <i>hours</i>
Commission- Employee	<i>commissionRate * grossSales</i>	commission employee: <i>firstName lastName</i> social security number: <i>SSN</i> gross sales: <i>grossSales</i> commission rate: <i>commissionRate</i>
BasePlus- Commission- Employee	<i>baseSalary + (commissionRate * grossSales)</i>	base salaried commission employee: <i>firstName lastName</i> social security number: <i>SSN</i> gross sales: <i>grossSales</i> commission rate: <i>commissionRate</i> base salary: <i>baseSalary</i>

Fig. 12.3 | Polymorphic interface for the Employee hierarchy classes.

```
// Fig. 12.4: Punonjes.cs
// Klasa abstrakte bazë: Punonjes
public abstract class Punonjes
{
    public string Emri { get; }
    public string Mbiemri { get; }
    public string NumriPersonal { get; }

    // konstruktori me tre parametra
    public Punonjes(string emri, string mbiemri,
        string numriPersonal)
    {
        Emri = emri;
        Mbiemri = mbiemri;
        NumriPersonal = numriPersonal;
    }

    // ktheje paraqitjen string të objektit Punonjes,
    // duke i përdorur properti-t
    public override string ToString() => $"{Emri} {Mbiemri}\n" +
        $"numri personal: {NumriPersonal}";

    // metodë abstrakte e mbishkruar nga klasat e derivuara
    public abstract decimal Fitimet(); // nuk ka implementim këtu
}
```

```

// Fig. 12.5: PunonjesMePage.cs
// Klasa PunonjesMePage që e zgjeron klasën Punonjes.
using System;

public class PunonjesMePage : Punonjes
{
    private decimal pagaJavore;

    // konstruktori me katër parametra
    public PunonjesMePage(string emri, string mbiemri,
        string numriPersonal, decimal pagaJavore)
        : base(emri, mbiemri, numriPersonal)
    {
        PagaJavore = pagaJavore; // validej pagën përmes properti-t
    }

    // properti që e merr (gets) dhe e cakton (sets)
    // pagën e punonjësit me pagë
    public decimal PagaJavore
    {
        get
        {
            return pagaJavore;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(PagaJavore)} duhet të jetë >= 0");
            }

            pagaJavore = value;
        }
    }

    // llogarit fitimet;
    // mbishkruaje metodën abstrakte Fitimet në klasën Punonjës
    public override decimal Fitimet() => PagaJavore;

    // ktheje paraqitjen string të objektit PunonjesMePage
    public override string ToString() =>
        $"punonjësi me pagë: {base.ToString()}\n" +
        $"paga javore: {PagaJavore:C}";
}

```



```
// Fig. 12.6: PunonjesMeOre.cs
// Klasa PunonjesMeOre që e zgjeron klasën Punonjes.
using System;

public class PunonjesMeOre : Punonjes
{
    private decimal cmimiIOres; // cmimiIOres (për orë)
    private decimal oret; // orët e punuara brenda javës

    // konstruktori me pesë parametra
    public PunonjesMeOre(string emri, string mbiemri,
        string numriPersonal, decimal cmimiIOres,
        decimal oretEPunuara)
        : base(emri, mbiemri, numriPersonal)
    {
        CmimiIOres = cmimiIOres; // validoje parametrin cmimiIOres
        Oret = oretEPunuara; // validoji orët e punuara
    }

    // properti që e merr (gets) dhe e cakton (sets)
    // çmimin e orës për punonjës
    public decimal CmimiIOres
    {
        get
        {
            return cmimiIOres;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(CmimiIOres)} duhet të jetë >= 0");
            }

            cmimiIOres = value;
        }
    }

    // properti që e merr (gets) dhe e cakton (sets)
    // orët e punuara për punonjës
    public decimal Oret
    {
        get
        {
            return oret;
        }
        set
        {
            if (value < 0 || value > 168) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(Oret)} duhet të jetë >= 0 dhe <= 168");
            }

            oret = value;
        }
    }
}
```

```

// llogariti fitimet;
// mbishkruaje metodën abstrakte Fitimet në klasën Punonjës
public override decimal Fitimet()
{
    if (Oret <= 40) // nuk ka punë jashtë orarit (overtime)
    {
        return CmimiIOres * Oret;
    }
    else
    {
        return (40 * CmimiIOres) + ((Oret - 40) * CmimiIOres * 1.5M);
    }
}

// ktheje paraqitjen string të objektit PunonjesMeOre
public override string ToString() =>
    $"punonjësi me orë: {base.ToString()}\n" +
    $"çmimi i orës: {CmimiIOres:C}\nnorë të punuara: {Oret:F2}";
}

```

// Fig. 12.7: PunonjesMeKomision.cs
 // Klasa PunonjesMeKomision që e zgjeron klasën Punonjes.
 using System;

```

public class PunonjesMeKomision : Punonjes
{
    private decimal shitjetBruto; // shitjet javore bruto
    private decimal normaEKomisionit; // përqindja e komisionit

    // konstruktori me pesë parametra
    public PunonjesMeKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEKomisionit)
        : base(emri, mbiemri, numriPersonal)
    {
        ShitjetBruto = shitjetBruto; // i validon shitjet bruto
        NormaEKomisionit = normaEKomisionit; // e validon normën e komisionit
    }

    // properti që i merr (gets) dhe i cakton (sets)
    // shitjet bruto të punonjësit
    public decimal ShitjetBruto
    {
        get
        {
            return shitjetBruto;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(ShitjetBruto)} duhet të jetë >= 0");
            }

            shitjetBruto = value;
        }
    }
}

```

```
// properti që e merr (gets) dhe e cakton (sets)
// normën e komisionit për punonjës
public decimal NormaEKomisionit
{
    get
    {
        return normaEKomisionit;
    }
    set
    {
        if (value <= 0 || value >= 1) // validimi
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(NormaEKomisionit)} duhet të jetë > 0 and < 1");
        }

        normaEKomisionit = value;
    }
}

// llogariti fitimet;
// mbishkruaje metodën abstrakte Fitimet në klasën Punonjës
public override decimal Fitimet() => NormaEKomisionit * ShitjetBruto;

// ktheje paraqitjen string të objektit PunonjesMeKomision
public override string ToString() =>
    $"punonjësi me komision: {base.ToString()}\n" +
    $"shitjet bruto: {ShitjetBruto:C}\n" +
    $"norma e komisionit: {NormaEKomisionit:F2}";
}
```

```
// Fig. 12.8: PunonjesMePageDheKomision.cs
// Klasa PunonjesMePageDheKomision që e zgjeron klasën Punonjes.
using System;

public class PunonjesMePageDheKomision : PunonjesMeKomision
{
    private decimal pagaBaze; // paga bazë për javë

    // konstruktori me gjashtë parametra
    public PunonjesMePageDheKomision(string emri, string mbiemri,
        string numriPersonal, decimal shitjetBruto,
        decimal normaEKomisionit, decimal pagaBaze)
        : base(emri, mbiemri, numriPersonal,
            shitjetBruto, normaEKomisionit)
    {
        PagaBaze = pagaBaze; // e validon pagën bazë
    }

    // properti që e merr (gets) dhe e cakton (sets)
    // pagën bazë për objektin PunonjesMePageDheKomision
    public decimal PagaBaze
    {
        get
        {
            return pagaBaze;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(PagaBaze)} duhet të jetë >= 0");
            }

            pagaBaze = value;
        }
    }

    // llogarit fitimet
    public override decimal Fitimet() => PagaBaze + base.Fitimet();

    // ktheje paraqitjen string të objektit PunonjesMePageDheKomision
    public override string ToString() =>
        $"[me pagë bazë] {base.ToString()}\npaga bazë: {PagaBaze:C}";
}

```

```
// Fig. 12.9: TestISistemitePagave.cs
// Aplikacion testues i hierarkisë së punonjësve.
using System;
using System.Collections.Generic;

class TestISistemitePagave
{
    static void Main()
    {
        // krijoji objektet e klasëve të derivuara
        var punonjesiMePage = new PunonjesMePage("John", "Smith",
            "111-11-1111", 800.00M);
        var punonjesiMeOre = new PunonjesMeOre("Karen", "Price",
            "222-22-2222", 16.75M, 40.00M);
        var punonjesiMeKomision = new PunonjesMeKomision("Sue", "Jones",
            "333-33-3333", 10000.00M, .06M);
        var punonjesiMePageDheKomision =
            new PunonjesMePageDheKomision("Bob", "Lewis",
            "444-44-4444", 5000.00M, .04M, 300.00M);

        Console.WriteLine("Punonjësit e procesuar individualisht:\n");

        Console.WriteLine($"{punonjesiMePage}\ni fitoi: " +
            $"{punonjesiMePage.Fitimet():C}\n");
        Console.WriteLine(
            $"{punonjesiMeOre}\ni fitoi: {punonjesiMeOre.Fitimet():C}\n");
        Console.WriteLine($"{punonjesiMeKomision}\ni fitoi: " +
            $"{punonjesiMeKomision.Fitimet():C}\n");
        Console.WriteLine($"{punonjesiMePageDheKomision}\ni fitoi: " +
            $"{punonjesiMePageDheKomision.Fitimet():C}\n");

        // krijoje një List<Punonjes> dhe inicializojë me objekte punonjës
        var punonjesit = new List<Punonjes>() {punonjesiMePage,
            punonjesiMeOre, punonjesiMeKomision, punonjesiMePageDheKomision};

        Console.WriteLine("Punonjësit e procesuar polimorfikisht:\n");

        // procesojë gjenerikisht (në mënyrë të përgjithshme)
        // secilin element të punonjesit
        foreach (var punonjesiAktual in punonjesit)
        {
            Console.WriteLine(punonjesiAktual); // e thërret ToString-un

            // përcaktoje se a është elementi PunonjesMePageDheKomision
            if (punonjesiAktual is PunonjesMePageDheKomision)
            {
                // downcast-oje (shndërroje teposhtë) referencën e Punonjes-it
                // në referencë të PunonjesMePageDheKomision
                var punonjesi = (PunonjesMePageDheKomision)punonjesiAktual;

                punonjesi.PagaBaze *= 1.10M;
                Console.WriteLine("paga e re bazë me 10% rritje është: " +
                    $"{punonjesi.PagaBaze:C}");
            }

            Console.WriteLine($"i fitoi: {punonjesiAktual.Fitimet():C}\n");
        }
    }
}
```

```
// merre emrin e tipit të secilit objekt te punonjesit
for (int j = 0; j < punonjesit.Count; j++)
{
    Console.WriteLine(
        $"Punonjësi {j} është {punonjesit[j].GetType()}");
}
Console.ReadKey();
}
```

Punonjësit e procesuar individualisht:

punonjësi me pagë: John Smith
numri personal: 111-11-1111
paga javore: £800.00
i fitoi: £800.00

punonjësi me orë: Karen Price
numri personal: 222-22-2222
çmimi i orës: £16.75
orë të punuara: 40.00
i fitoi: £670.00

punonjësi me komision: Sue Jones
numri personal: 333-33-3333
shitjet bruto: £10,000.00
norma e komisionit: 0.06
i fitoi: £600.00

[me pagë bazë] punonjësi me komision: Bob Lewis
numri personal: 444-44-4444
shitjet bruto: £5,000.00
norma e komisionit: 0.04
paga bazë: £300.00
i fitoi: £500.00

Punonjësit e procesuar polimorfikisht:

punonjësi me pagë: John Smith
numri personal: 111-11-1111
paga javore: £800.00
i fitoi: £800.00

punonjësi me orë: Karen Price
numri personal: 222-22-2222
çmimi i orës: £16.75
orë të punuara: 40.00
i fitoi: £670.00

punonjësi me komision: Sue Jones
numri personal: 333-33-3333
shitjet bruto: £10,000.00
norma e komisionit: 0.06
i fitoi: £600.00

[me pagë bazë] punonjësi me komision: Bob Lewis
numri personal: 444-44-4444
shitjet bruto: £5,000.00
norma e komisionit: 0.04
paga bazë: £300.00
paga e re bazë me 10% rritje është: £330.00
i fitoi: £530.00

Punonjësi 0 është PunonjesMePage
Punonjësi 1 është PunonjesMeOre
Punonjësi 2 është PunonjesMeKomision
Punonjësi 3 është PunonjesMePageDheKomision

- Duke e deklaruar një metodë **abstract**-e, tregojmë se çdo klasë e derivuar konkrete duhet ta ofrojë një implementim të duhur.
- Për të gjitha thirrjet e metodës **virtual**-e duhet të vendoset gjatë kohës së ekzekutimit, bazuar në tipin e objektit të cilit i referohet variabla e tipit referencë. Ky proces njihet si lidhje dinamike (dynamic binding) apo lidhje e vonshme (late binding).
- Operatori **is** e përcakton se a përputhet tipi i objektit në operandin e majtë me tipin e specifikuar nga operandi i djathtë dhe kthen **true** nëse ata dy kanë relacion **is-a** (*është-një*).
- Operatori **as** e kryen një downcast (shndërrim teposhtë në hierarki të klasave) që e kthen një referencë në objektin e duhur nëse downcast-i është i suksesshëm dhe kthen **null** nëse downcast-i dështon.
- Një referencë e klasës bazë mund të përdoret për t'i thirrur veç metodat e deklaruara në klasën bazë. Nëse një aplikacioni i duhet ta kryejë një operacion specifik të klasës së derivuar në një objekt të klasës së derivuar të referencuar nga një variabël e klasës bazë, aplikacioni duhet së pari ta downcast-ojë referencën e klasës bazë në një referencë të klasës së derivuar.
- Çdo objekt e di tipin e vet dhe mund t'i qaset këtij informacioni përmes metodës **GetType**, të cilën e trashëgojnë të gjitha klasat nga klasa **object**.
- Caktimi i referencës klasës bazë – një variable të klasës së derivuar nuk lejohet pa cast (shndërrim) eksplisit apo pa e përdorur operatorin **as**. Operatori **is** mund të përdoret për t'u siguruar se një cast (shndërrim) i tillë bëhet vetëm nëse objekti është objekt i klasës së derivuar.

12.6 Metodatat dhe klasat sealed (të vulosura)

- Një metodë që është deklaruar **sealed** në një klasë bazë nuk mund të mbishkruhet në një klasë të derivuar.
- Një klasë që është deklaruar **sealed** nuk mund të jetë klasë bazë (d.m.th.. një klasë nuk mund ta zgjerojë një klasë **sealed** [të vulosur]). Të gjitha metodat në një klasë **sealed** janë **sealed** në mënyrë implicite.

12.7 Studim i rastit: Krijimi dhe përdorimi i interfejsave (ndërfaqeve)

- Interfejsat i definojnë dhe i standardizojnë mënyrat në të cilat gjërat – siç janë njerëzit dhe sistemet – mund të ndërveprojnë me njëri-tjetrin.
- Një deklarinim i interfejsit fillon me fjalën kyçe **interface** dhe mund të përmbajë vetëm metoda, properti, indeksa (indeksues) dhe event-a (ngjarje) **abstract**-e. Ata nuk specifikojnë kurrfarë detalesh të implementimit, siç janë deklarinimet e metodave konkrete.
- Secili interfejs mund ta zgjerojë (extend) një apo më shumë interfejsa për ta krijuar një interfejs më të elaboruar (të përpunuar) të cilin mund ta implementojnë klasat e tjera.
- Një klasë duhet të specifikojë që ajo e implementon interfejsin duke e listuar (renditur) atë pas dypikëshit (:) në deklarinimin e klasës.
- Një klasë që e implementon një interfejs por nuk i implementon të gjithë anëtarët e interfejsit duhet të deklarohet **abstract**-e dhe ta përmbajë një deklarinim abstrakt të secilit anëtarë të paimplementuar të interfejsit.
- UML e shpreh relacionin ndërmjet një klase dhe një interfejsi përmes një realizimi. Një klasë thuhet se e “realizon” apo e implementon, një interfejs.
- Për ta implementuar më shumë se një interfejs, përdoreni një listë të ndarë me presje të emrave të interfejsave pas dypikëshit (:) në deklarinimin e klasës.
- Trashëgimia dhe ndërfaqet janë të ngjashme në implementimin e tyre të relacionit **is-a (është-një)**. Një objekt i një klase që e implementon një interfejs mund të mendohet si një objekt i atij tipi të interfejsit.
- Të gjitha metodat e klasës **object** mund të thirren duke e përdorur një referencë të një tipi të interfejsit – referenca i referohet një objekti, dhe të gjitha objektet i trashëgojnë metodat e klasës **object**.

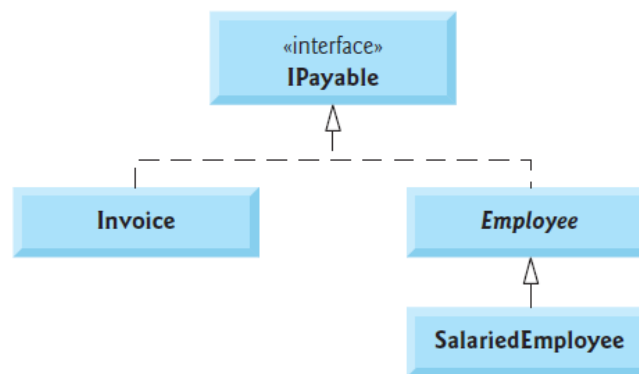


Fig. 12.10 | IPayable interface and class hierarchy UML class diagram.

```

// Fig. 12.11: IEPagueshme.cs
// Deklarimi i interfejsit (ndërfaqes) IEPagueshme
public interface IEPagueshme
{
    decimal MerreSasineEPageses(); // llogarite pagesën; nuk ka implementim
}
  
```

```

// Fig. 12.12: Fature.cs
// Klasa Fature e implementon interfejsin (ndërfaqen) IEPagueshme.
using System;

public class Fature : IEPagueshme
{
    public string NumriIPjeses { get; }
    public string PershkrimiIPjeses { get; }
    private int sasias;
    private decimal cmimiPerArtikull;

    // konstruktor me katër parametra
    public Fature(string numriIPjeses, string pershkrimiIPjeses, int sasias,
        decimal cmimiPerArtikull)
    {
        NumriIPjeses = numriIPjeses;
        PershkrimiIPjeses = pershkrimiIPjeses;
        Sasias = sasias; // validoje sasinë
        CmimiPerArtikull = cmimiPerArtikull; // validoje çmimin për artikull
    }
}
  
```

```

// properti që e merr (gets) dhe e cakton (sets)
// sasinë në faturë
public int Sasia
{
    get
    {
        return sasia;
    }
    set
    {
        if (value < 0) // validimi
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(Sasia)} duhet të jetë >= 0");
        }

        sasia = value;
    }
}

// properti që e merr (gets) dhe e cakton (sets)
// çmimin për artikull
public decimal CmimiPerArtikull
{
    get
    {
        return cmimiPerArtikull;
    }
    set
    {
        if (value < 0) // validimi
        {
            throw new ArgumentOutOfRangeException(nameof(value),
                value, $"{nameof(CmimiPerArtikull)} duhet të jetë >= 0");
        }

        cmimiPerArtikull = value;
    }
}

// ktheje paraqitjen në string të objektit Faturë
public override string ToString() =>
    $"fatura:\nnumri i pjesës: {NumriIPjeses} ({PershkrimiIPjeses})\n" +
    $"sasia: {Sasia}\nçmimi për artikull: {CmimiPerArtikull:C}";

// metoda e kërkuar për ta përmbushur kontratën me interfejsin IEPagueshme
public decimal MerreSasineEPageses() => Sasia * CmimiPerArtikull;
}

```

```
// Fig. 12.13: Punonjes.cs
// Klasa abstrakte bazë Punonjës që e implementon interfejsin IEPagueshme.
public abstract class Punonjes : IEPagueshme
{
    public string Emri { get; }
    public string Mbiemri { get; }
    public string NumriPersonal { get; }

    // konstruktor me tre parametra
    public Punonjes(string emri, string mbiemri,
        string numriPersonal)
    {
        Emri = emri;
        Mbiemri = mbiemri;
        NumriPersonal = numriPersonal;
    }

    // ktheje paraqitjen në string të objektit Punonjës,
    // duke i përdorur properti-t
    public override string ToString() => $"{Emri} {Mbiemri}\n" +
        $"numri personal: {NumriPersonal}";

    // metoda abstrakte e mbishkruar nga klasat e derivuara
    public abstract decimal Fitimet(); // nuk ka implementim këtu

    // implementimi i metodës MerreSasineEPageses ia mundëson
    // tërë hierarkisë së klasave Punonjës ta përdorin
    // në një aplikacion që i proceson objektet IEPagueshme
    public decimal MerreSasineEPageses() => Fitimet();
}
```

```

// Fig. 12.5: PunonjesMePage.cs
// Klasa PunonjesMePage që e zgjeron klasën Punonjes.
using System;

public class PunonjesMePage : Punonjes
{
    private decimal pagaJavore;

    // konstruktori me katër parametra
    public PunonjesMePage(string emri, string mbiemri,
        string numriPersonal, decimal pagaJavore)
        : base(emri, mbiemri, numriPersonal)
    {
        PagaJavore = pagaJavore; // validej pagën përmes properti-t
    }

    // properti që e merr (gets) dhe e cakton (sets)
    // pagën e punonjësit me pagë
    public decimal PagaJavore
    {
        get
        {
            return pagaJavore;
        }
        set
        {
            if (value < 0) // validimi
            {
                throw new ArgumentOutOfRangeException(nameof(value),
                    value, $"{nameof(PagaJavore)} duhet të jetë >= 0");
            }

            pagaJavore = value;
        }
    }

    // llogarit fitimet;
    // mbishkruaje metodën abstrakte Fitimet në klasën Punonjës
    public override decimal Fitimet() => PagaJavore;

    // ktheje paraqitjen string të objektit PunonjesMePage
    public override string ToString() =>
        $"punonjësi me pagë: {base.ToString()}\n" +
        $"paga javore: {PagaJavore:C}";
}

```

```
// Fig. 12.14: TestIInterfejsitEPagueshme.cs
// E teston interfejsin IEPagueshme me klasa të ndryshme.
using System;
using System.Collections.Generic;

class TestIInterfejsitEPagueshme
{
    static void Main()
    {
        // krijoje një List<IEPagueshme> dhe inicializojë
        // me katër objekte të klasave që e implementojnë
        // interfejsin IEPagueshme
        var objektetEPagueshme = new List<IEPagueshme>() {
            new Fature("01234", "karrige", 2, 375.00M),
            new Fature("56789", "gomë", 4, 79.95M),
            new PunonjesMePage("John", "Smith", "111-11-1111", 800.00M),
            new PunonjesMePage("Lisa", "Barnes", "888-88-8888", 1200.00M)};

        Console.WriteLine(
            "Faturat dhe Punonjësit të procesuar polimorfikisht:\n");

        // procesoje gjenerikisht (në mënyrë të përgjithshme)
        // secilin element në objektetEPagueshme
        foreach (var iPagueshmi in objektetEPagueshme)
        {
            // afishojë (shfaqë në dalje) objektin iPagueshme
            // dhe sasinë e tij përkatëse të pagesës
            Console.WriteLine($"{iPagueshmi}");
            Console.WriteLine(
                $"për pagesë: {iPagueshmi.MerreSasineEPageses():C}\n");
        }
        Console.ReadKey();
    }
}
```

```
Faturat dhe Punonjësit të procesuar polimorfikisht:

fatura:
numri i pjesës: 01234 (karrige)
sasia: 2
çmimi për artikull: £375.00
për pagesë: £750.00

fatura:
numri i pjesës: 56789 (gomë)
sasia: 4
çmimi për artikull: £79.95
për pagesë: £319.80

punonjësi me pagë: John Smith
numri personal: 111-11-1111
paga javore: £800.00
për pagesë: £800.00

punonjësi me pagë: Lisa Barnes
numri personal: 888-88-8888
paga javore: £1,200.00
për pagesë: £1,200.00
```

Interface	Description
<code>IComparable</code>	C# contains several comparison operators (e.g., <code><</code> , <code><=</code> , <code>></code> , <code>>=</code> , <code>==</code> , <code>!=</code>) that allow you to compare simple-type values. Section 10.13 showed that you can overload these operators for your own types. Interface <code>IComparable</code> can be used to allow objects of a class that implements the interface to be compared to one another. The interface contains one method, <code>CompareTo</code> , which compares the object that calls the method to the object passed as an argument. Classes must implement <code>CompareTo</code> to return a value indicating whether the object on which it's invoked is less than (negative integer return value), equal to (0 return value) or greater than (positive integer return value) the object passed as an argument, using any criteria you specify. For example, if class <code>Employee</code> implements <code>IComparable</code> , its <code>CompareTo</code> method could compare <code>Employee</code> objects by their earnings amounts. Interface <code>IComparable</code> is commonly used for ordering objects in a collection such as an array. We use <code>IComparable</code> in Chapter 20, Generics, and Chapter 21, Generic Collections; Functional Programming with LINQ/PLINQ.
<code>IComponent</code>	Implemented by any class that represents a component, including Graphical User Interface (GUI) controls (such as buttons or labels). Interface <code>IComponent</code> defines the behaviors that components must implement. We discuss <code>IComponent</code> and many GUI controls that implement this interface in Chapter 14, Graphical User Interfaces with Windows Forms: Part 1, and Chapter 15, Graphical User Interfaces with Windows Forms: Part 2.

Fig. 12.15 | Common interfaces of the .NET Framework Class Library. (Part 1 of 2.)

Interface	Description
IDisposable	Implemented by classes that must provide an explicit mechanism for <i>releasing</i> resources. Some resources can be used by only one program at a time. In addition, some resources, such as files on disk, are unmanaged resources that, unlike memory, cannot be released by the garbage collector. Classes that implement interface <code>IDisposable</code> provide a <code>Dispose</code> method that can be called to explicitly release resources that are explicitly associated with an object. We discuss <code>IDisposable</code> briefly in Chapter 13, <i>Exception Handling: A Deeper Look</i> . You can learn more about this interface at http://msdn.microsoft.com/library/system.idisposable . The MSDN article <i>Implementing a Dispose Method</i> at http://msdn.microsoft.com/library/fs2xkftw discusses the proper implementation of this interface in your classes.
IEnumerator	Used for iterating through the elements of a <i>collection</i> (such as an array or a <code>List</code>) one element at a time—the <code>foreach</code> statement uses an <code>IEnumerator</code> object to iterate through elements. Interface <code>IEnumerator</code> contains method <code>MoveNext</code> to move to the next element in a collection, method <code>Reset</code> to move to the position before the first element and property <code>Current</code> to return the object at the current location. We use <code>IEnumerator</code> in Chapter 21. All <code>IEnumerable</code> objects (Chapter 9) provide a <code>GetEnumerator</code> method that returns an <code>IEnumerator</code> object.

Fig. 12.15 | Common interfaces of the .NET Framework Class Library. (Part 2 of 2.)

Bibliografia

Deitel, Paul, and Harvey Deitel. *Visual C# how to program*. Pearson, 2016.

Online

<https://rbunjaku.wordpress.com/>

Licensimi

Ky publikim lëshohet nën licensën



Attribution-NonCommercial-ShareAlike

CC BY-NC-SA

This license lets others remix, tweak, and build upon your work non-commercially, as long as they credit you and license their new creations under the identical terms.

që nënkupton se i lejon të tjerët ta shkarkojnë dhe ta shpërndajnë te të tjerët, ta ndryshojnë punën në mënyrë jokomerciale, përderisa e përmendin burimin dhe i licencojnë krijimet e reja nën terma identike. Për më tepër, shih:

<http://creativecommons.org/licenses/>