

Paul Deitel dhe Harvey Deitel

Hyrje në programim

(përmbledhje të kapitujve dhe ushtrime)

botimi i dytë – me ushtrime e figura

përzgjedhi dhe shqipëroi Ridvan Bunjaku

tetor 2022 – janar 2023, maj 2023

Përmbajtja

2. Hyrje në Visual Studio dhe Programim Vizual (Pamor)	5
2.1 Hyrje.....	5
2.2 Vështrim i përgjithshëm i IDE-së Visual Studio Community	5
2.3 Shiriti i menysë (Menu bar-i) dhe Shiriti i mjeteve (Toolbar-i)	8
2.4 Navigimi nëpër pjesët e ndryshme të IDE-së Visual Studio	12
2.5 Menyja e Ndhmës dhe Context-Sensitive Help (Ndihma sipas kontekstit)	17
2.6 Visual Programming (Programimi vizual): Krijimi i një aplikacioni të thjeshtë që shfaq tekst dhe imazh (image, imazh, foto)	17
3. Hyrje në programimin e aplikacioneve me C#.....	26
3.2 Aplikacion i thjeshtë: Shfaqja e një rreshti të tekstit	26
3.2.1 Komentet.....	26
3.2.2 Direktiva <i>using</i>	27
3.2.3 Rreshtat e zbrazët dhe hapësirat e zbrazëta (whitespace)	27
3.2.4 Deklarimi i klasës.....	27
3.2.5 Metoda <i>Main</i>	29
3.2.6 Shfaqja e një rreshti me tekst.....	29
3.3 Krijimi i një aplikacioni të thjeshtë në Visual Studio.....	30
3.3.1 Krijimi i <i>Console App</i> (aplikacionit për konsolë)	30
3.3.3 Shkrimi i kodit dhe përdorja e <i>IntelliSense</i>	32
3.3.5 Gabimet, mesazhet e gabimeve dhe Error List Window (Dritarja Error List, Lista e gabimeve)	34
3.4 Ndryshimi (modifikimi) i aplikacionit tuaj të thjeshtë në C#	35
3.4.1 Shfaqja e një rreshti të tekstit me shumë urdhra	35
3.4.2 Paraqitja e shumë rreshtave të tekstit me një urdhër të vetëm	36
3.5 Interpolimi (ndërfutja) e stringut	38
3.6 Një aplikacion tjetër në C#: Mbledhja e numrave të plotë	39
3.6.1 Deklarimi i ndryshores së tipit <i>int</i>	40
3.6.3 Kërkimi i të dhënave hyrëse nga përdoruesi.....	40
3.6.6 Mbledhja e dy numrave	41
3.7 Koncepte të memories	41
3.8 Aritmetika	42
3.8.3 Rregullat e përparësisë së operatorëve (operator precedence)	43
3.9 Marrja e vendimeve: Barazia dhe Operatorët Relacionalë.....	44

4. Hyrje në klasa, objekte, metoda dhe stringje	48
4.1 Hyrje.....	48
4.2 Testimi i një klase <i>Llogari (Account)</i>	48
4.2.1 Instancimi i një Objekti – Fjala kyçe <i>new</i> dhe konstruktorët	49
4.2.2 Thirrja e metodës <i>GetName (MerreEmrin)</i> të klasës <i>Account (LlogariEBankes)</i>	50
4.2.3 Futja e një emri nga përdoruesi	50
4.2.4 Thirrja e metodës <i>Set</i> të klasës	50
4.3 Klasa me ndryshore të instancës dhe metoda <i>Set</i> dhe <i>Get</i>	50
4.3.1 Deklarimi i klasës.....	51
4.3.2 Fjala kyçe <i>class</i> dhe trupi i klasës	51
4.3.3 Variabla (ndryshorja) e instancës e tipit <i>string</i>	52
4.3.4 Metoda <i>SetName</i>	53
4.3.5 Metoda <i>GetName</i>	53
4.3.6 Modifikuesit (ndryshuesit) e qasjes <i>private</i> dhe <i>public</i>	54
4.3.7 Diagrami i klasës në UML (Unified Modeling Language – Gjuha e unifikuar e modelimit)	55
4.4 Krijimi, kompajlimi dhe ekzekutimi i një projekti me dy klasa	56
4.5-6 Inxhinierimi i softuerit me metodat <i>Set</i> dhe <i>Get</i> , <i>Property</i>	57
4.6.2 Klasa me variabël të instancës dhe një properti	59
4.6.3 Diagrami UML i klasës me një properti	60
4.7 Properti-t e implementuara automatikisht (auto-implemented)	60
4.8 Inicializimi i objekteve me konstruktorë	61
4.8.1 Deklarimi i konstruktorit për inicializim specifik të objektit	62
4.8.2 Inicializimi i objekteve kur krijohen	62
4.9 Klasa me variabël <i>decimal</i> të instancës.....	64
4.9.2 Klasa që krijon dhe përdor objekte.....	66
5. Zhvillimi i algoritmeve dhe Urdhrat e kontrollit: Pjesa 1	69
5.2 Algoritmet.....	69
5.3 Pseudokodi	69
5.4 Strukturat e kontrollit.....	70
5.4.1 Struktura sekuencë.....	71
5.4.2 Urdhrat e përzgjedhjes	71
5.4.3 Urdhrat e përsëritjes (iterimit)	72
5.4.4 Përmbledhje e urdhreve të kontrollit.....	72
5.4.5 Urdhri <i>if</i> me një përzgjedhje	72

5.6 Urdhri <i>if...else</i> me përzgjedhje të dyfishtë.....	73
5.7 Klasa <i>Student</i> : Urdhrat e ndërfutur <i>if...else</i>	74
5.8 Urdhri i përsëritjes (iterimit) <i>while</i>	76
5.9 Formulimi i algoritmeve: Përsëritja (iterimi) i kontrolluar me numërues (counter-controlled iteration)	77
5.10 Formulimi i algoritmeve: Përsëritja (iterimi) i kontrolluar me vlerë vëzhguese (sentinel-controlled iteration).....	79
5.11 Formulimi i algoritmeve: Urdhrat e kontrollit që janë të ndërfutur (nested).....	82
Detajimi i dytë i plotë i pseudokodit	87
5.12 Operatorët e përbërë të caktimit të vlerës (Compound assignment operators).....	91
5.13 Operatorët e rritjes dhe zbritjes së vlerës (increment and decrement operators)	91
5.13.3 Përparësia dhe asociativiteti i (shoqërueshmëria e) operatorëve.....	94
5.14 Tipet e thjeshta	94
6. Urdhrat e kontrollit: Pjesa 2.....	95
6.2 Bazat e përsëritjes së (iterimit të) kontrolluar nga numëruesi (Counter-Controlled Iteration)	95
6.3 Urdhri i përsëritjes (iterimit) <i>for</i>	96
6.5 Aplikacion: Mbledhja e numrave çiftë	98
6.6 Aplikacion: Llogaritjet e interesit të përbërë	99
6.7 Urdhri i përsëritjes (iterimit) <i>do...while</i>	101
6.8 Urdhri <i>switch</i> me përzgjedhje të shumëfishtë	103
6.9 Studimi i rastit Klasa <i>PoliseEVeturës</i> : stringjet në urdhra <i>switch</i>	107
6.10 Urdhrat <i>break</i> dhe <i>continue</i>	109
6.11 Operatorët logjikë	111
6.12 Përmbledhje e Programimit-Të-Strukturuar	115
Për botimin.....	119
Bibliografia	119
Online	119
Licensimi	119

2. Hyrje në Visual Studio dhe Programim Vizual (Pamor)

2.1 Hyrje

- **Visual Studio** është **Integrated Development Environment** (IDE - Mjedis/Ambient/Rrethinë e Integruar e Zhvillimit) nga Microsoft për **krijimin, ekzekutimin dhe debugim** (debugging, korrigjimin e gabimeve) të aplikacioneve të shkruara në një sërë gjuhësh të programimit në .NET (dot net).
- Krijimi i aplikacioneve të thjeshta me **drag and drop**, pra duke i bartur/zvarritur dhe lëshuar blloqet e paradefinuara ndërtuese në vendin e tyre quhet **zhvillim vizual/pamor i aplikacioneve** (visual app development).

2.2 Vështrim i përgjithshëm i IDE-së Visual Studio Community

- **Projekti** është grup i fajllave të ndërlidhur që e përbëjnë një aplikacion.
- Visual Studio i organizon aplikacionet nëpër projekte dhe **solucione** (zgjidhje, **solutions**). Një solucion mund të përmbajë një apo më shumë projekte.
- **Dialogjet (dialogs)** janë dritare që e lehtësojnë komunikimin përdorues-kompjuter.
- Visual Studio ofron **shabllone (templates)** për llojet e projekteve që mund t'i krijosh.
- Një **Form (formë / formular)** e paraqet dritaren kryesore të aplikacionit të llojit **Windows Forms** që je duke e krijuar.
- Bashkërisht (kolektivisht), Forma dhe kontrollat e përbëjnë **ndërfaqen grafike të përdoruesit** (graphical user interface, GUI) për aplikacionin, e cila është pjesa **vizuale** (pamore) e aplikacionit me të cilën përdoruesi **ndërvepron** (interacts).

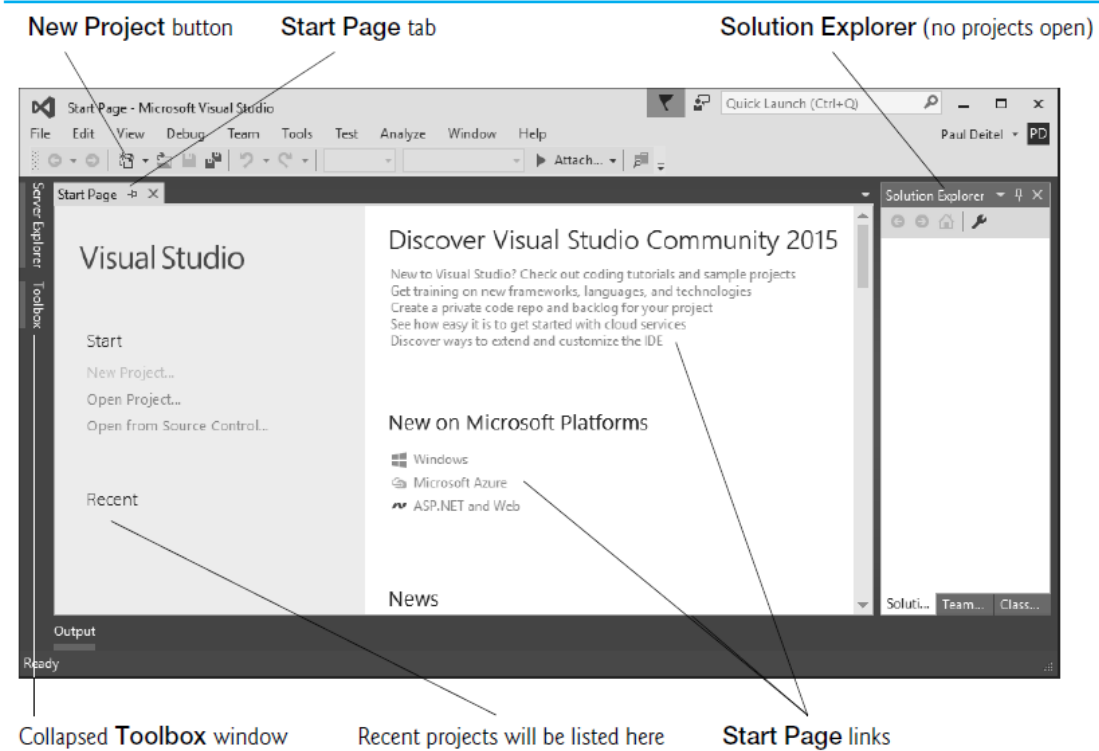


Fig. 2.1 | Start Page in Visual Studio Community 2015.

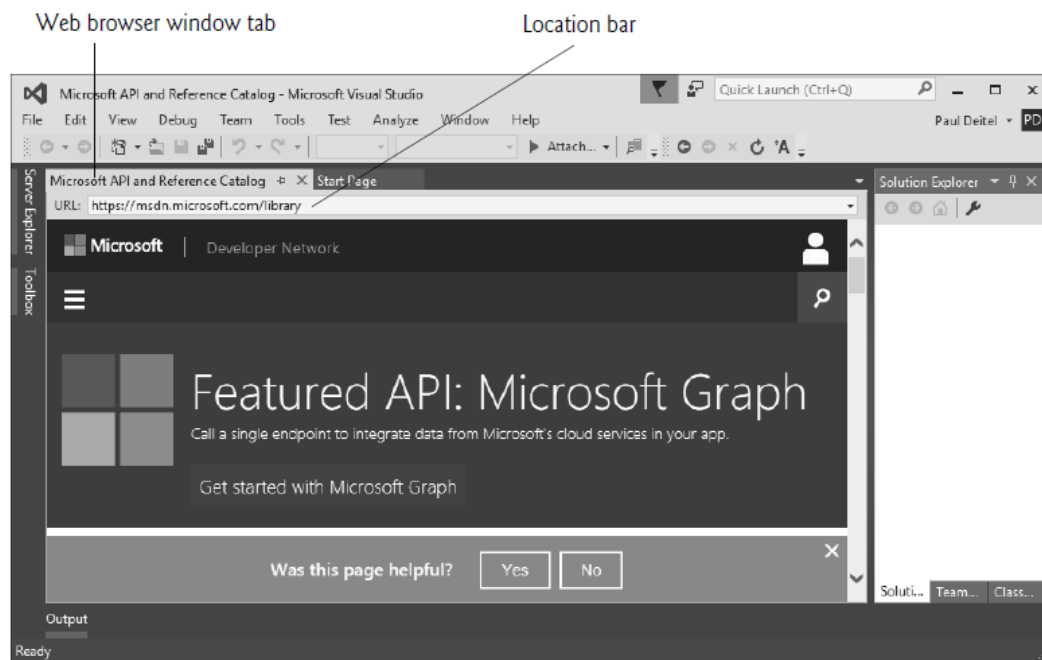


Fig. 2.2 | MSDN Library web page in Visual Studio.

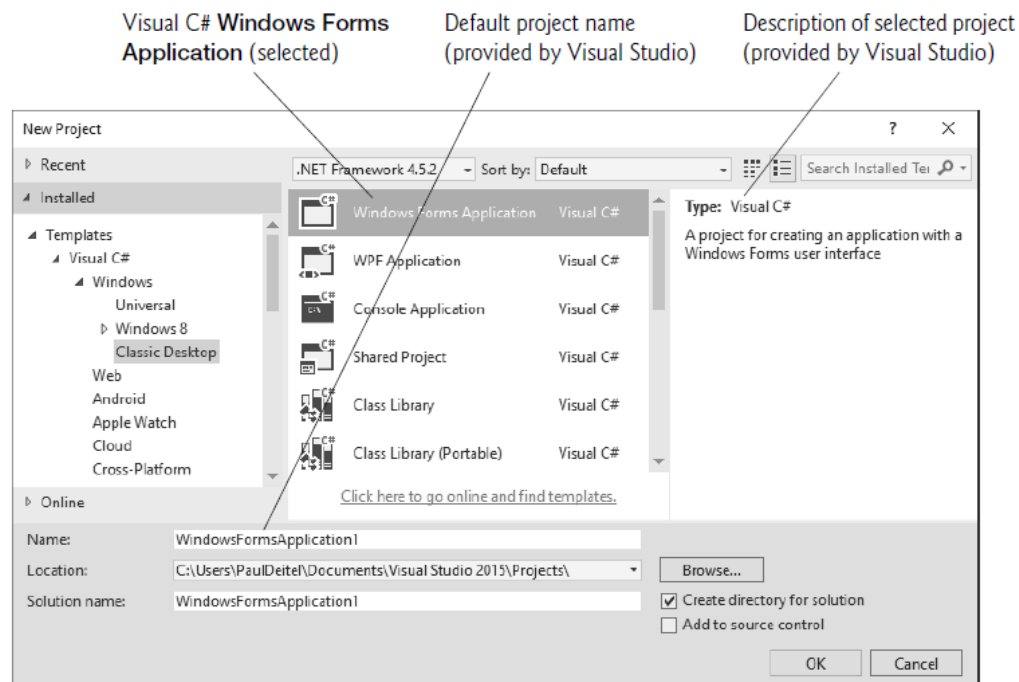


Fig. 2.3 | New Project dialog.

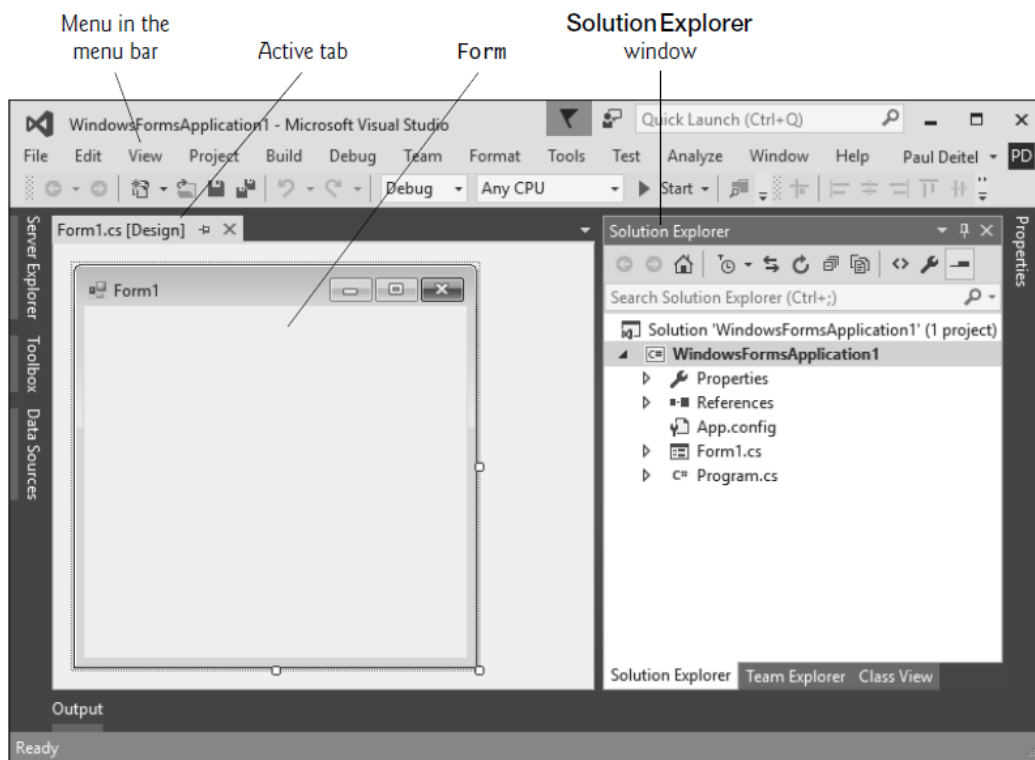


Fig. 2.4 | Design view of the IDE.

2.3 Shiriti i menysë (Menu bar-i) dhe Shiriti i mjeteve (Toolbar-i)

- Komandat për menaxhimin e IDE-së dhe për zhvillimin, mirëmbajtjen dhe ekzekutimin e aplikacioneve janë nëpër menytë, që ndodhen në **shiritin e menysë (menu bar)**.
- Menytë përmbajnë grupe të komandave (**elemente të menysë – menu items**) që, kur zgjedhen, bëjnë që IDE të kryejë veprime (për shembull: hape një dritare, ruaje një fajll, printoje një fajll dhe ekzekutoje një aplikacion).
- **Tool tips (këshillat e mjeteve)** të ndihmojnë të njihesh me **features (veçoritë)** e IDE-së.

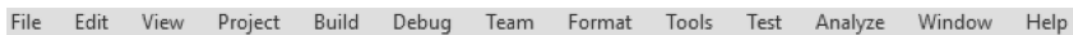


Fig. 2.5 | Visual Studio menu bar.

Menyja	Përmban komanda për
File (Fajllat)	Hapjen, mbyllhen, shtimin dhe ruajtjen e projekteve, si dhe printimin e shënimeve të projektit dhe daljen nga Visual Studio
Edit (Edito, Ndrysho)	Editimi i aplikacioneve si cut (preje), copy (kopjoje), paste (pastoje apo ngjite), undo (zhbëje), redo (ribëje), delete (fshije), find (gjeje) dhe select (zgjedhe).
View (Pamja, Shiko)	Shfaqja e dritareve të IDE (për shembull dritaret Solution Explorer (Eksploruesi i Zgjidhjes) , Toolbox (Kutia e veglave apo Instrumentari) , Properties (Vetitë)).
Project (Projekti)	Menaxhimi i projekteve dhe i fajllave të tyre.
Build (Ndërtoje, Ndërtimi, Kompajlimi)	Shndërrimi i aplikacionit tuaj në program të ekzekutueshëm.
Debug (Debagimi, Heqja e defekteve)	Kompajlimi, debugimi (heqja e defekteve) (domethënë, identifikimi dhe korrigjimi i problemeve në aplikacione) dhe ekzekutimi i aplikacioneve.
Team (Ekipi)	Lidhja me një Team Foundation Server – që përdoret nga ekipet e zhvillimit që zakonisht kanë shumë njerëz që punojnë në aplikacionin e njëjtë.
Format (Formati, Formatizo, Formatizimi)	Rregullimi dhe ndryshimi (modifikimi) i kontrollave të një Form -e. Menyja Format shfaqet vetëm kur një komponentë e GUI (Graphical User Interface, Ndërfaqes Grafike të Përdoruesit) është e zgjedhur në Design view (në pamjen Design , pra pamjen e dizajnit).
Tools (Veglat, Instrumentet)	Vendi për qasje në mjete (instrumente) dhe opsione shtesë të IDE për ta përshtatur dhe rregulluar (custom-izuar) IDE-në.
Test (Testi, Testo)	Kryerja e tipeve të ndryshme të testimit të automatizuar në aplikacionin tuaj.
Analyze (Analizo)	Alokimi (gjetja se ku janë) dhe raportimi i shkeljeve të .NET Framework Design Guidelines (Udhëzimeve të Dizajnit të Kornizës .NET) (https://msdn.microsoft.com/library/ms229042).
Window (Dritarja)	Fshehja, hapja, mbyllja dhe shfaqja e dritareve të IDE.
Help (Ndihma)	Qasja në veçoritë e ndihmës të IDE-së.

Fig. 2.6 | Përmbledhje e menyve të Visual Studio-s që shfaqen kur një **Form**-ë është në **Design** view (pamjen e dizajnit).

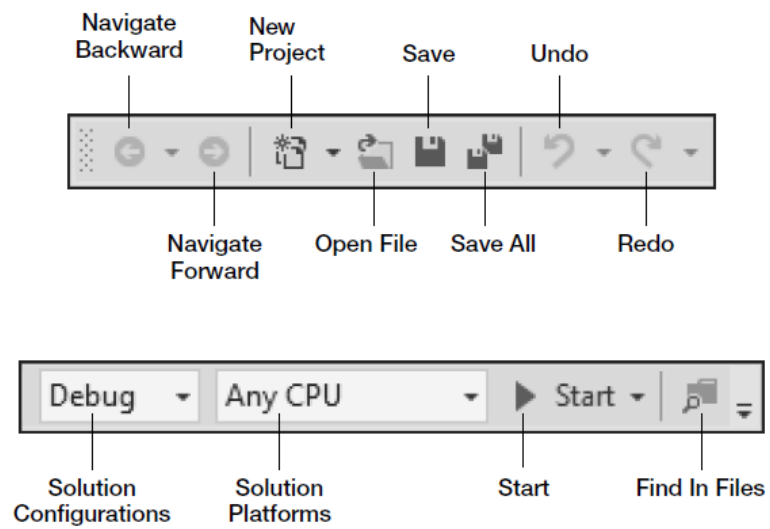


Fig. 2.7 | Standard Visual Studio toolbar.



Fig. 2.8 | List of toolbars that can be added to the top of the IDE.

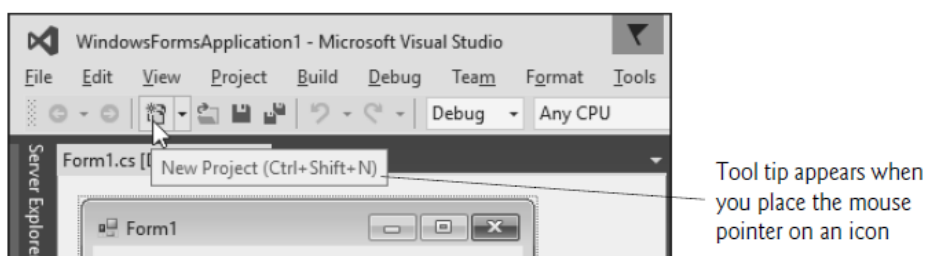


Fig. 2.9 | Tool tip for the New Project button.

2.4 Navigimi nëpër pjesët e ndryshme të IDE-së Visual Studio

- Dritarja **Solution Explorer** (**Eksploruesi i Zgjidhjes**) i liston të gjithë fajllat në solucion.
- **Toolbox** (**kutia e mjeteve**) i përmban kontrollat për customizimin (përshtatjen, specifikimin apo personalizimin) e **Formave**.
- Duke e përdorur **visual app development** (**zhvillimin vizual/pamor të aplikacioneve**), mund t'i vendosësh kontrollat e paradefinuara në **Formë** në vend se ta shkruash kodin vetë.
- Klikimi i emrit të dritares **auto-hide** (që fshehet vetë) e hap atë dritare. Klikimi i emrit përsëri e fsheh atë. Për ta "fiksuar" ("pin down") një dritare (domethënë, për ta çaktivizuar auto-hide-in – fshehjen automatike), kliko në ikonën e saj **pin**.
- Dritarja **Properties** (Vetitë) i shfaq vetitë për një **Formë**, kontrollë apo fajll (në **Design view** – pamjen e dizajnit). Properties (vetitë, proprietit) janë informacion për një **Formë** apo kontrollë, siç është **size** (madhësia), **color** (ngjyra) dhe **position** (pozita apo pozicioni). Dritarja **Properties** të lejon t'i modifikosh (ndryshosh) **Format** dhe kontrollat në mënyrë vizuale (pamore) pa shkruar kod.
- Secila kontrollë e ka grupin e saj të **propertive** (vetive). Kolona (shtylla, column) e majtë e dritares **Properties** i tregon emrat e propertive dhe shtylla e djathtë i shfaq vlerat e propertive. Toolbar-i (shiriti i mjeteve) i kësaj dritareje i përmban opsionet për organizimin e propertive sipas alfabetit kur zgjedhet ikona **Alphabetical** (Alfabetike), apo sipas kategorive (për shembull, **Appearance** (Paraqitja), **Behavior** (Sjellja), **Design** (Dizajni) etj.) kur zgjedhet ikona **Categorized** (Të kategorizuara).

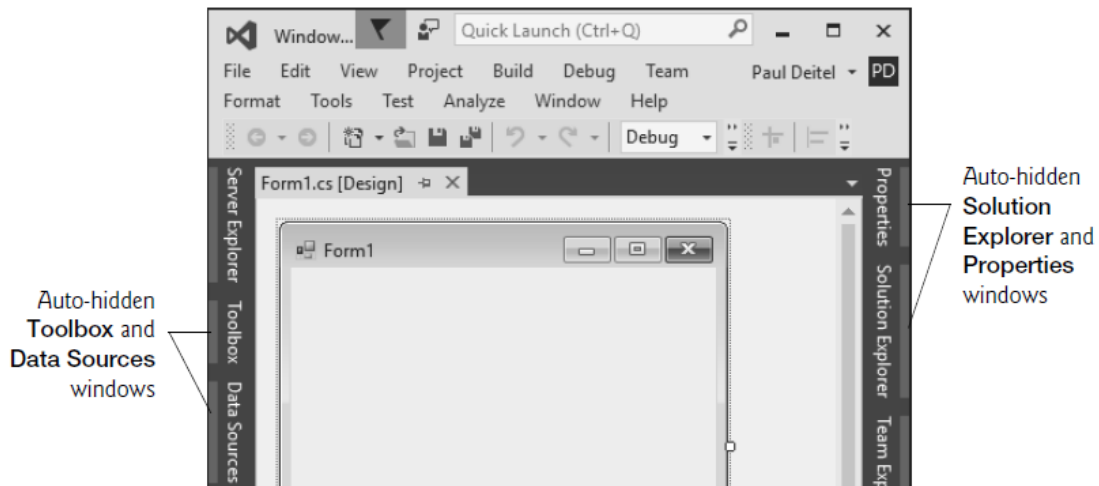


Fig. 2.10 | Auto-hide feature demonstration.

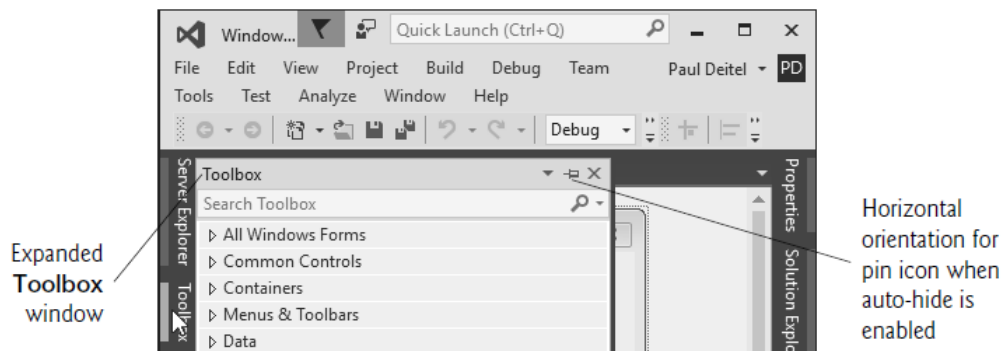


Fig. 2.11 | Displaying the hidden Toolbox window when auto-hide is enabled.

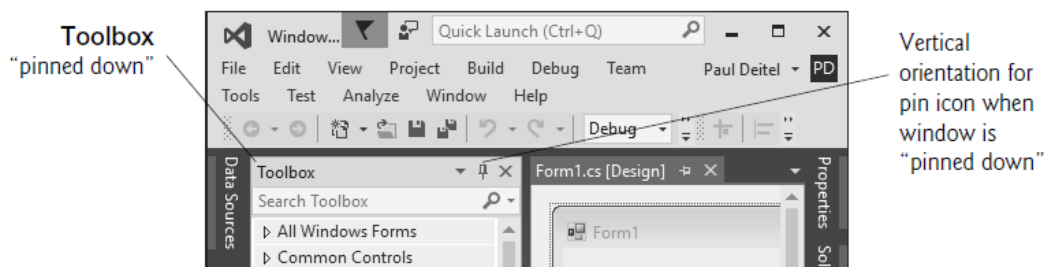


Fig. 2.12 | Disabling auto-hide—"pinning down" a window.

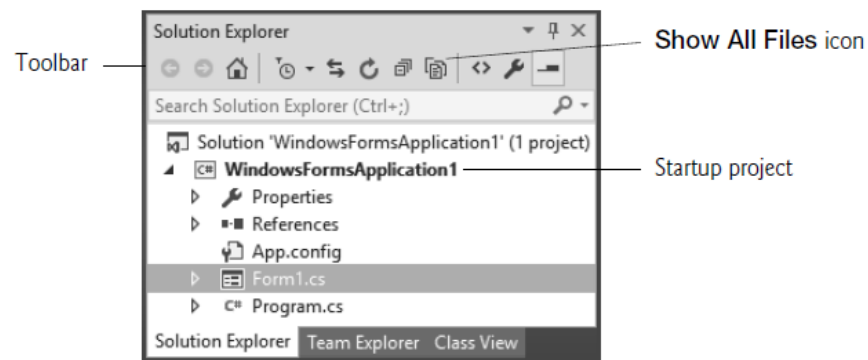


Fig. 2.13 | Solution Explorer window showing the `WindowsFormsApplication1` project.

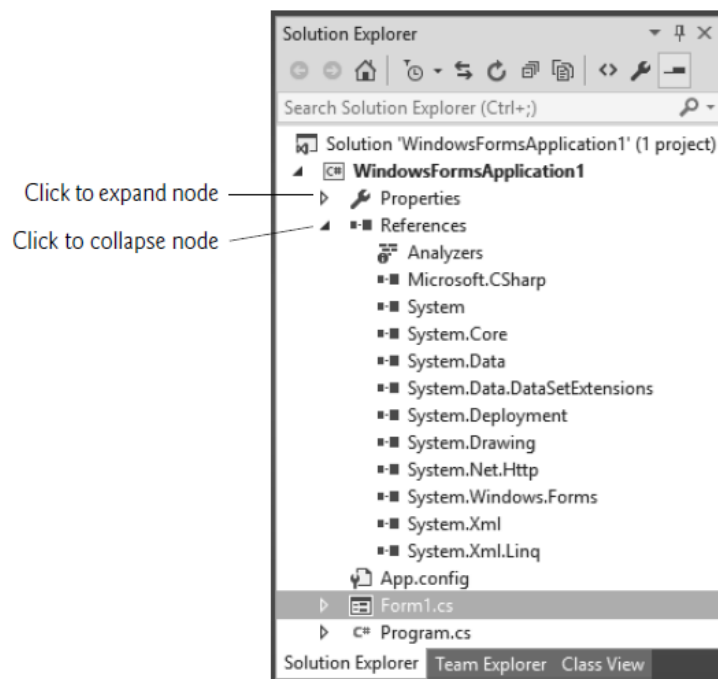


Fig. 2.14 | Solution Explorer with the `References` node expanded.

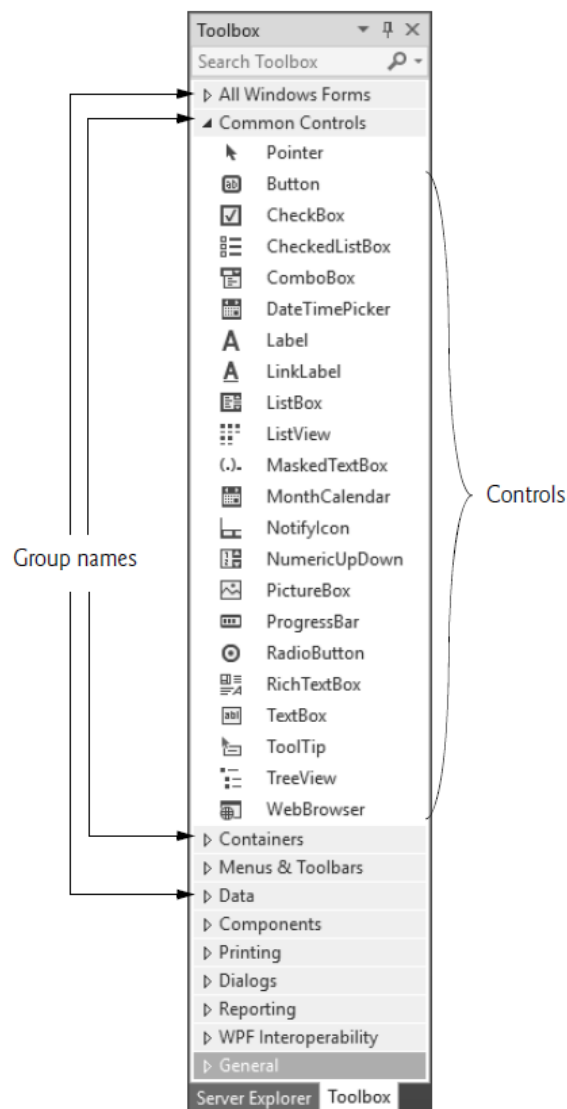


Fig. 2.15 | Toolbox window displaying controls for the Common Controls group.

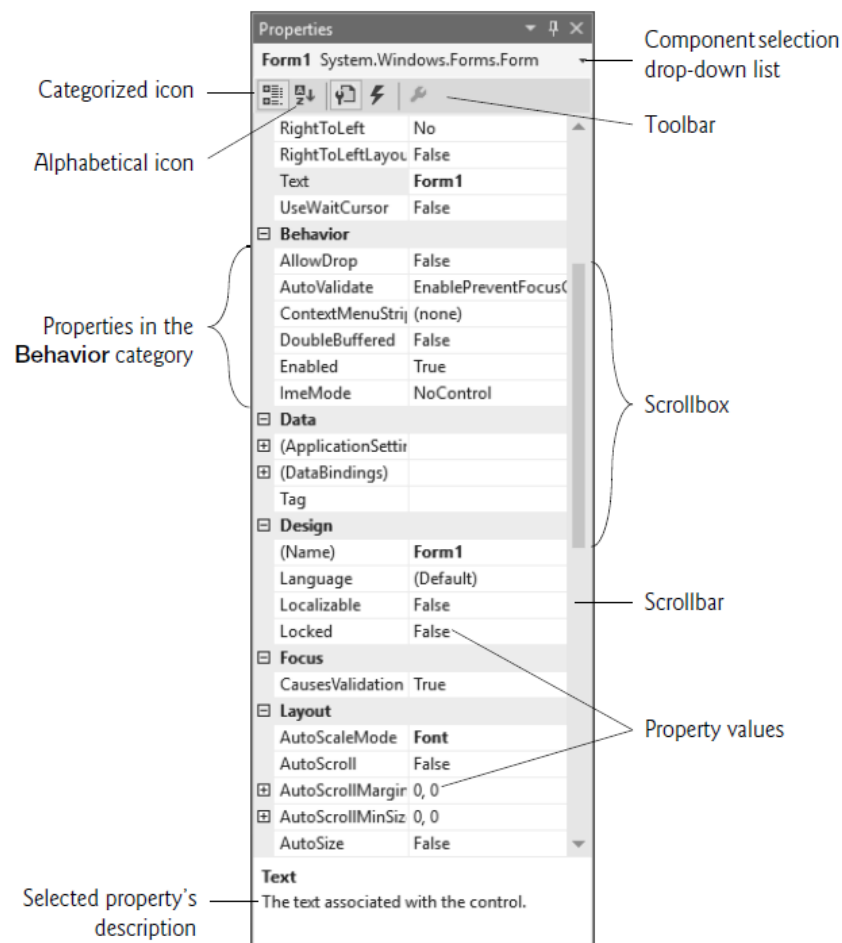


Fig. 2.16 | Properties window.

2.5 Menyja e Ndhmës dhe Context-Sensitive Help (Ndhma sipas kontekstit)

- Dokumentacioni i gjerë i ndihmës është i disponueshëm përmes menysë **Help (Ndhmë)**.
- **Context-Sensitive Help** (Ndhma sipas kontekstit) e sjell një listë me artikuj përkatës të ndihmës. Për ta përdorur context-sensitive help, zgjedhe një element dhe shtype tastin **F1**.

2.6 Visual Programming (Programimi vizual): Krijimi i një aplikacioni të thjeshtë që shfaq tekst dhe imixh (image, imazh, foto)

- Visual C# app development (Zhvillimi vizual i aplikacioneve në C#) zakonisht e përfshin një kombinim të shkrimit të një pjese të kodit të aplikacionit dhe Visual Studio e gjeneron pjesën tjetër të kodit.
- Teksti që shfaqet në krye të **Formës** (title bar – shiriti i titullit) specifikohet në propertin **Text** të **Formës**.
- Për ta ndryshuar madhësinë e **Formës**, klikoje dhe tërhiqe një nga **enabled sizing handles** (dorezat e aktivizuara për madhësi) (katrorët e vegjël rreth **Formës**). Enabled sizing handles shfaqen si kuti të bardha.
- Properti **BackColor** e specifikon ngjyrën e sfondit (background color) të një **Forme**. Background color e **Formës** është ngjyra e paracaktuar e sfondit (default background color) për çfarëdo kontrole që i shtohet **Formës**.
- **Klikimi i dyfishtë (Double click-imi)** në çfarëdo **control icon** (ikonë të kontrollës) në **Toolbox** e vendos një kontrollë të atij lloji në **Formë**. Alternativisht, mund t'i tërheqësh kontrollat nga Toolbox-i dhe t'i lëshosh në **Formë**.
- Properti (vetia) **Text** e **Labelës** e përcakton tekstin (nëse ka) të cilin e shfaq Labela. Forma dhe Labela - secila e ka propertin e saj **Text**.
- Butoni **tripikësh (ellipsis, "...")**, kur klikohet, e shfaq një dialog që përmban opsione shtesë.

- Në dialogun **Font**, mund ta zgjedhësh fontin për tekstin në **ndërfaqen e përdoruesit (user interface)**.
- Property **TextAlign** e përcakton se si është rreshtuar teksti brenda kufijve të Label-ës.
- Kontrolla **PictureBox** i shfaq imazhet. Property **Image** e specifikon imazhin që do të shfaqet.
- Një aplikacion që është në **design mode** (modin apo modalitetin e dizajnit/projektimit) nuk është duke u ekzekutuar.
- Në **run mode** (modin apo modalitetin e ekzekutimit), aplikacioni është duke u ekzekutuar – mund të ndërveprosh veç me disa veçori të IDE-së (IDE features).
- Gjatë dizajnit të një aplikacioni vizualisht, emri i fajllit Visual C# shfaqet në tab-in e projektit, i pasuar nga fjala “Design” në kllapa të mesme: **[Design]**.
- Përfundoje ekzekutimin e një aplikacioni duke klikuar në close box (kutinë e mbylljes).

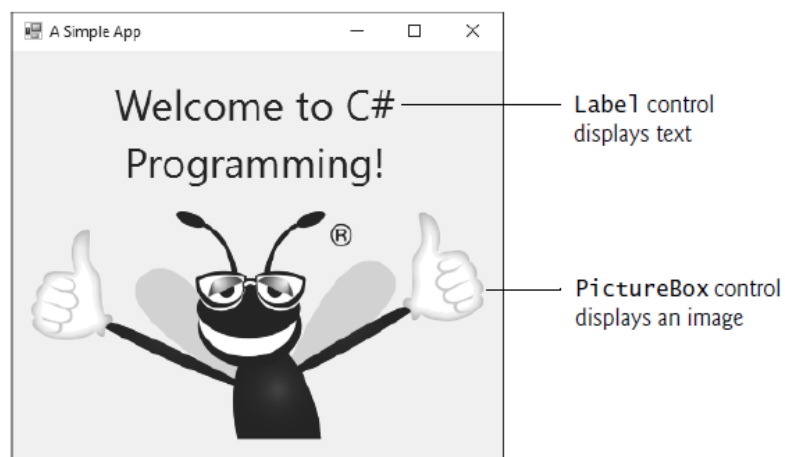
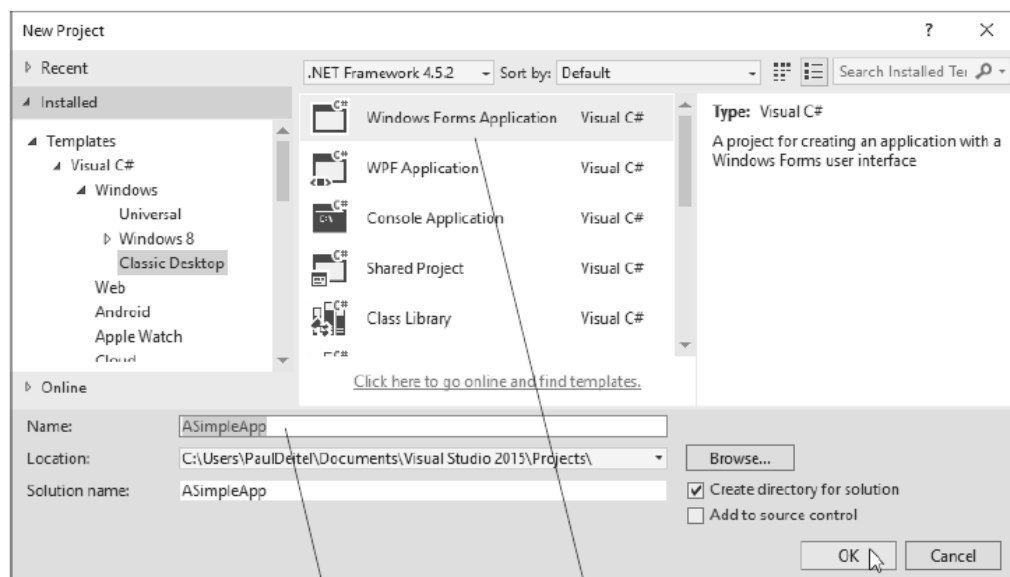


Fig. 2.17 | Simple app executing.



Type the project
name here

Select the **Windows Forms
Application** template

Fig. 2.18 | New Project dialog.

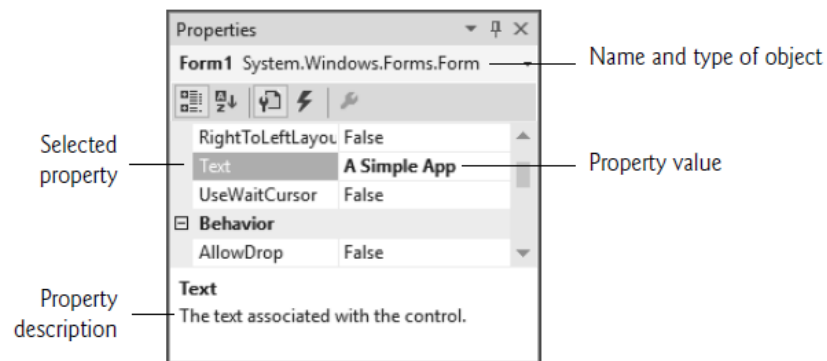


Fig. 2.19 | Setting the Form's Text property in the Properties window.

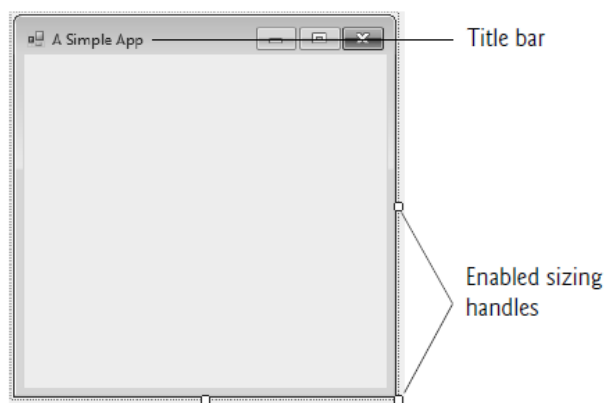


Fig. 2.20 | Form with updated title-bar text and enabled sizing handles.

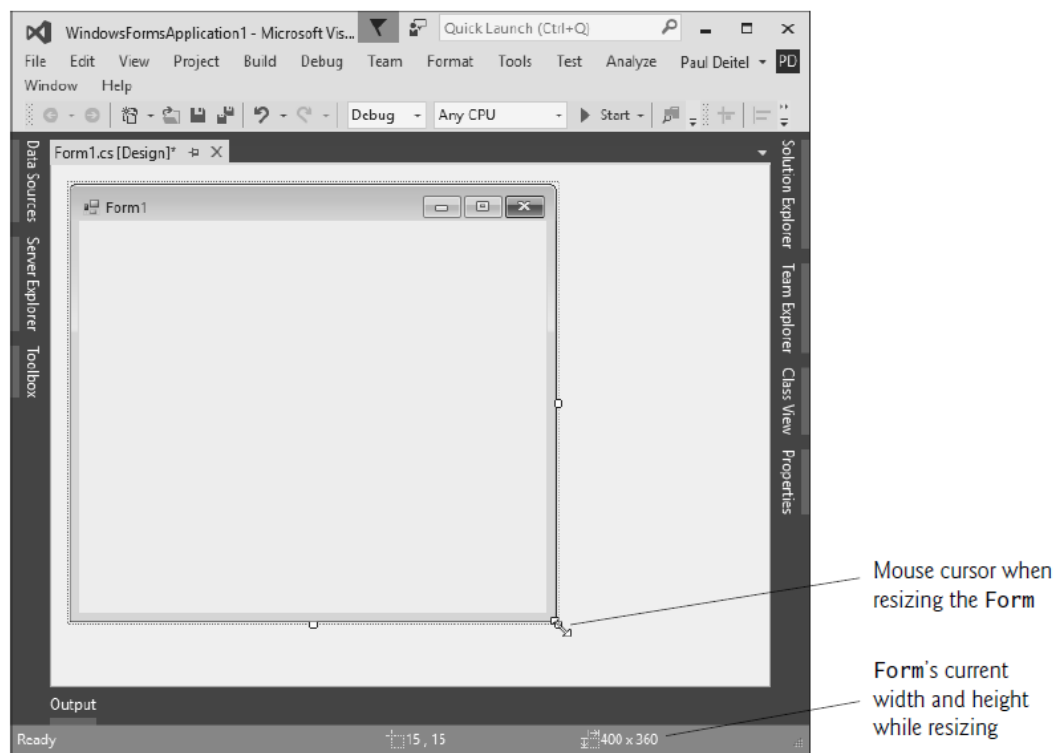


Fig. 2.21 | Resizing the Form.

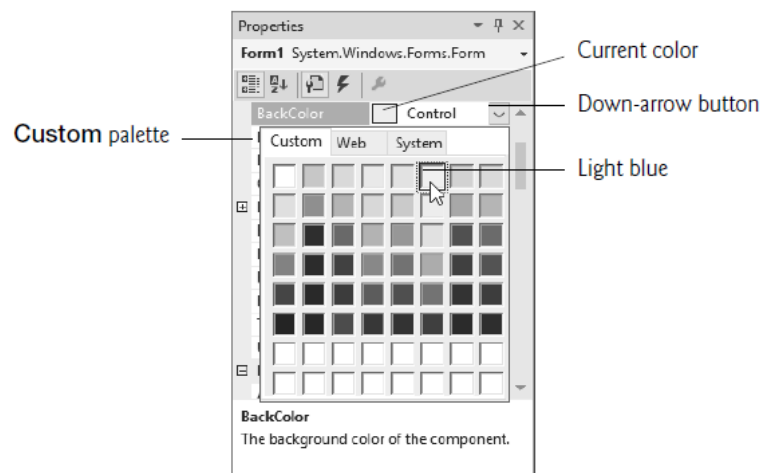


Fig. 2.22 | Changing the Form's BackColor property.

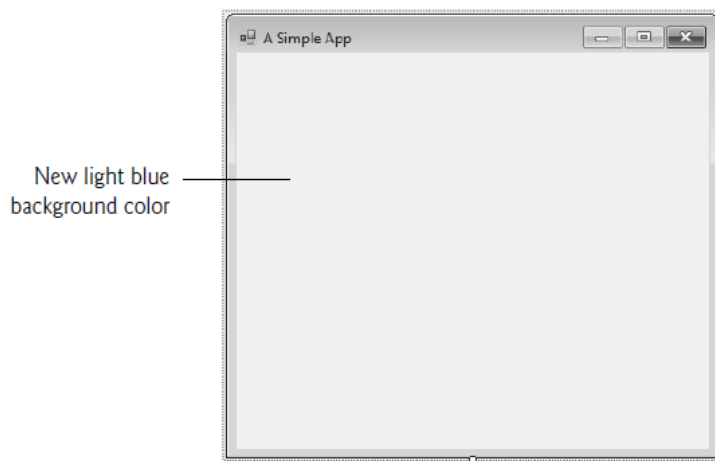


Fig. 2.23 | Form with new BackColor property applied.

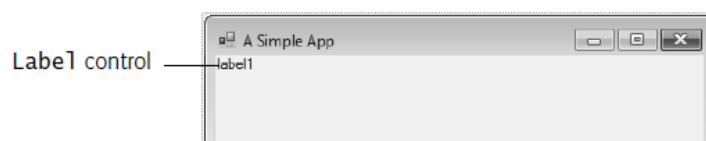


Fig. 2.24 | Adding a Label to the Form.

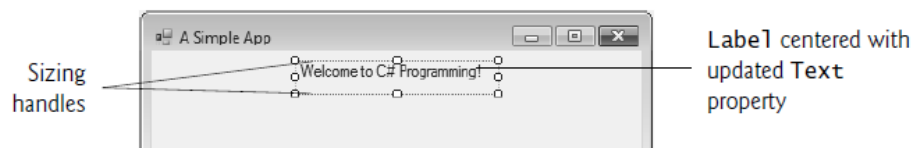


Fig. 2.25 | GUI after the Form and Label have been customized.

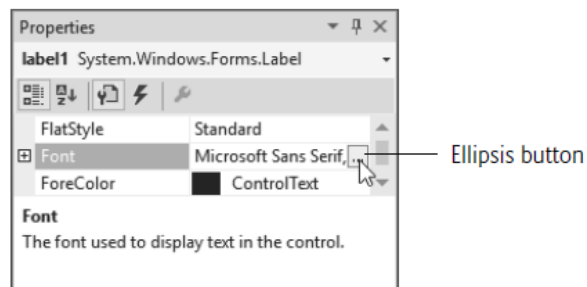


Fig. 2.26 | Properties window displaying the Label's Font property.

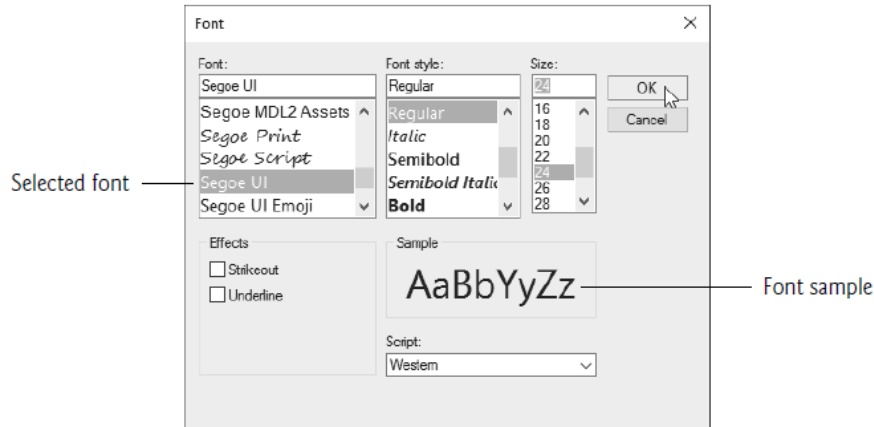


Fig. 2.27 | Font dialog for selecting fonts, styles and sizes.

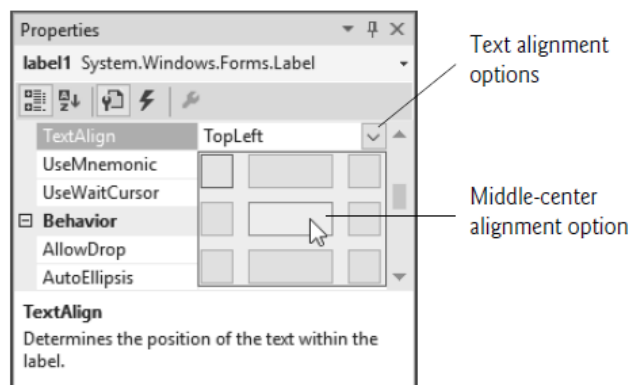


Fig. 2.28 | Centering the Label's text.

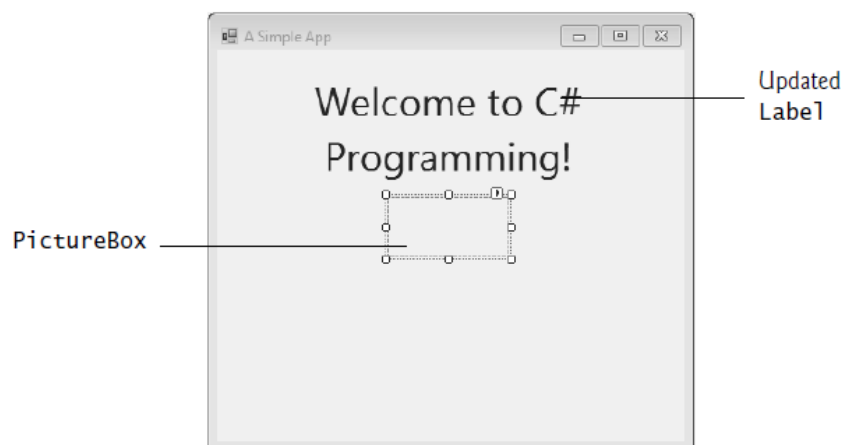


Fig. 2.29 | Inserting and aligning a PictureBox.

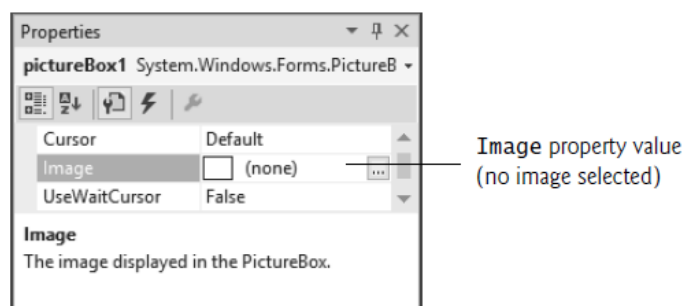


Fig. 2.30 | Image property of the PictureBox.

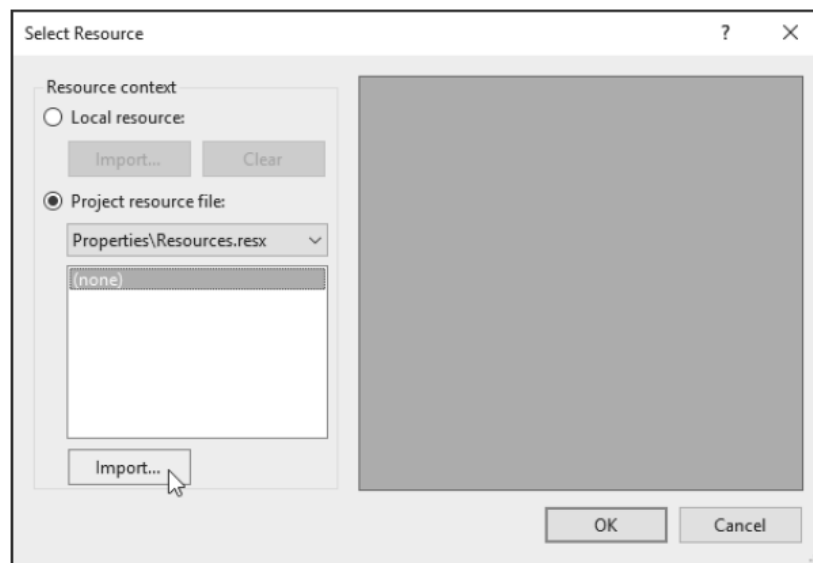


Fig. 2.31 | Select Resource dialog to select an image for the PictureBox.

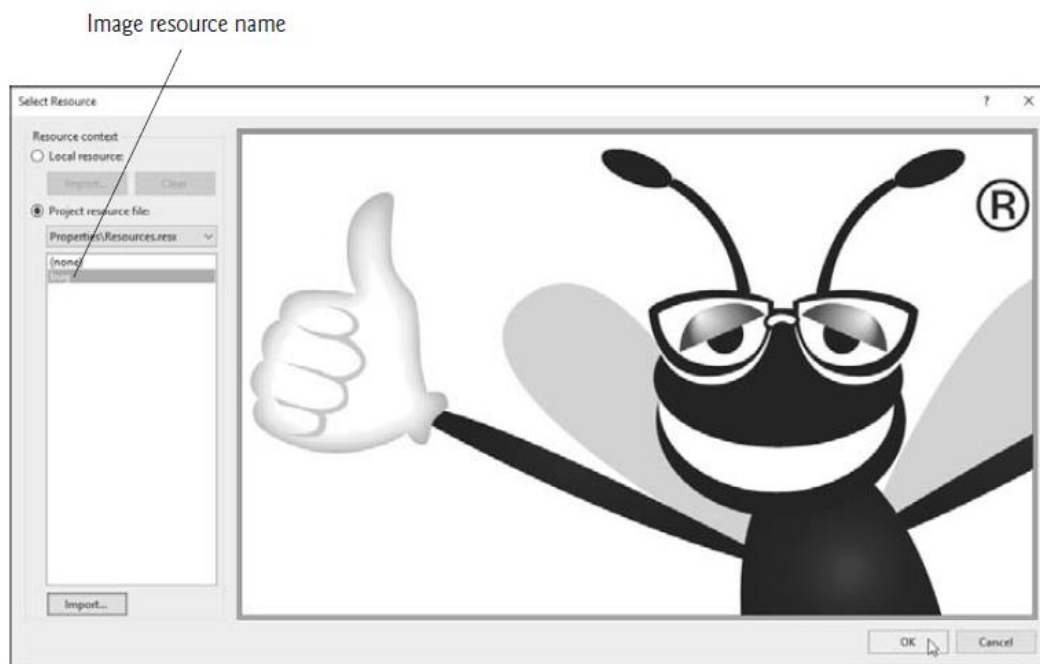


Fig. 2.32 | Select Resource dialog displaying a preview of selected image.

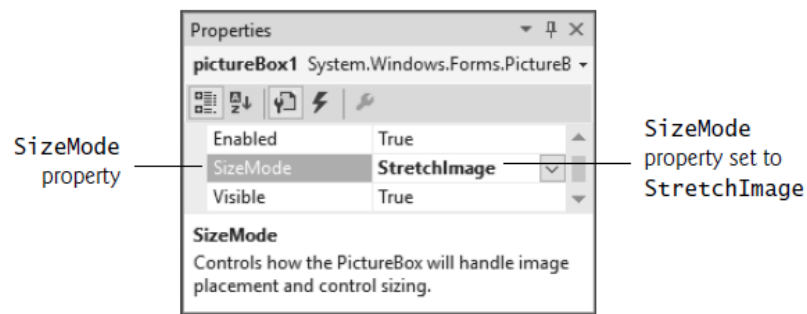


Fig. 2.33 | Scaling an image to the size of the PictureBox.

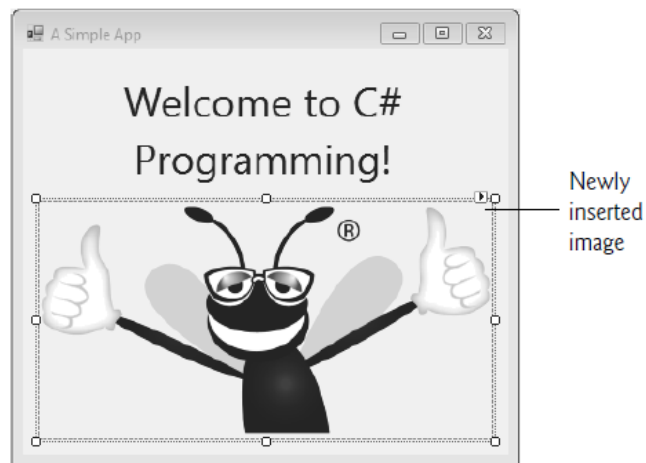


Fig. 2.34 | PictureBox displaying an image.

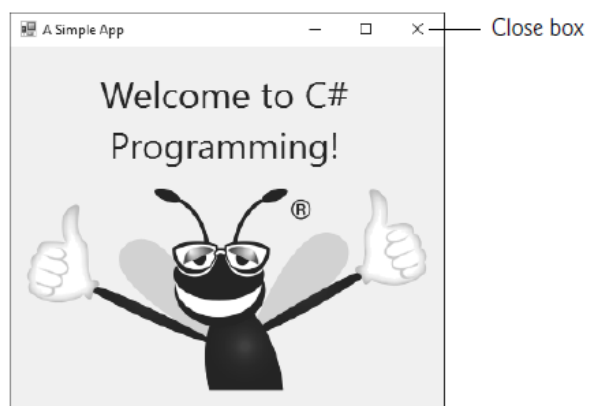


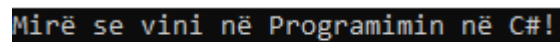
Fig. 2.35 | IDE in run mode, with the running app in the foreground.

3. Hyrje në programimin e aplikacioneve me C#

3.2 Aplikacion i thjeshtë: Shfaqja e një rreshti të tekstit

```
// Fig. 3.1: MireSeVini1.cs
// Aplikacion që shfaq tekst.
using System;

class MireSeVini1
{
    // Metoda Main e nis ekzekutimin e aplikacionit në C#
    static void Main()
    {
        Console.WriteLine("Mirë se vini në Programimin në C#!");
        Console.ReadKey();
    } // përfundojë Main-in
} // përfundojë klasën MireSeVini1
```



```
Mirë se vini në Programimin në C#!
```

3.2.1 Komentet

- Mund të vendosësh komente për t'i dokumentuar aplikacionet dhe për ta përmirësuar lexueshmërinë e tyre. Kompajleri i C# i injoron komentet.
- Komenti që fillon me // quhet koment një-rreshtor, sepse përfundon në fund të rreshtit në të cilin shfaqet.
- Komentet e kufizuara me /* dhe */ mund të shpërndahen nëpër disa rreshta.
- Sintaksa e gjuhës programuese i specifikon rregullat për krijimin e një aplikacioni të duhur në atë gjuhë.

3.2.2 Direktiva *using*

- Direktiva **using** i ndihmon kompajlerit ta gjejë një tip (type) që përdoret në një aplikacion.
- C# e ofron një grup të pasur të klasave të parafinuar që mund t'i ripërdorni në vend se ta "rishpikni rrotën". Këto klasa janë të grupuara nëpër **namespace**-a (hapësira të emrave) – koleksione të emëruara të klasave.
- Bashkërisht namespace-at e parafinuar të C# i quajmë **.NET Framework Class Library** (Libraria e klasave të Kornizës .NET).

3.2.3 Rreshtat e zbrazët dhe hapësirat e zbrazëta (whitespace)

- Mund t'i përdorni rreshtat e zbrazët dhe karakteret **space** (hapësirë) për t'i bërë aplikacionet më të lexueshme.
- Bashkërisht, rreshtat e zbrazët, karakteret **space** dhe karakteret **tab** njihen si **whitespace** (hapësira të zbrazëta).
- Karakteret space dhe tab-at njihen specifikisht si **karaktere whitespace** (për hapësira të zbrazëta).
- Hapësirat e zbrazëta injorohen nga kompajleri.

3.2.4 Deklarimi i klasës

- Çdo aplikacion në C# përbëhet nga të paktën një deklaram i klasës që definohet nga programuesi (që njihet edhe si **user-defined class** - **klasë e definuar nga përdoruesi**).
- **Keywords** (fjalët kyçe, kryesore) janë të rezervuara për përdorim nga C# dhe gjithmonë shkruhen **me të gjitha shkronjat të vogla (lowercase)**.
- Fjala kyçe **class** e paraqet një deklaram të klasës dhe pasohet menjëherë nga emri i klasës.

- **Me marrëveshje (convention)**, të gjithë emrat e klasave në C# nisin me shkronjë të madhe dhe e kanë të madhe shkronjën e parë të secilës fjalë që e përfshijnë (p.sh. SampleClassName apo EmërShembullIKlasës). Kjo marrëveshje njihet si **Pascal Case**.
- Emri i klasës në C# është identifikues (identifier) – seri e karaktereve që përbëhet nga shkronja, shifra dhe viza poshtë (underscores: _), që nuk fillon me shifër dhe nuk përmban hapësira.
- C# është case-sensitive – domethënë i dallon shkronjat e mëdha nga të voglat.
- **Trupi (body)** i secilit deklarinim të klasës kufizohet me kllapa gjarpërore: { dhe }.

Keywords and contextual keywords					
abstract	as	base	bool	break	byte
case	catch	char	checked	class	const
continue	decimal	default	delegate	do	double
else	enum	event	explicit	extern	false
finally	fixed	float	for	foreach	goto
if	implicit	in	int	interface	internal
is	lock	long	namespace	new	null
object	operator	out	override	params	private
protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string
struct	switch	this	throw	true	try
typeof	uint	ulong	unchecked	unsafe	ushort
using	virtual	void	volatile	while	
<i>Contextual Keywords</i>					
add	alias	ascending	async	await	by
descending	dynamic	equals	from	get	global
group	into	join	let	nameof	on
orderby	partial	remove	select	set	value
var	where	yield			

Fig. 3.2 | Keywords and contextual keywords.

3.2.5 Metoda *Main*

- Metoda **Main** është pika fillestare e çdo aplikacioni në C# dhe zakonisht definohet si:

```
static void Main()
```

- Metodat mund të kryejnë detyra (task-a, punë) dhe mund të kthejnë informacion kur i kryejnë detyrat e veta. Fjala kyçe **void** (bosh, i zbrazët) tregon se një metodë do ta kryejë një detyrë por nuk do të kthejë vlerë.

3.2.6 Shfaqja e një rreshti me tekst

- **Urdhrrat** (statements, formulimet) e udhëzojnë kompjuterin të kryejë veprime.
- Një sekuencë (varg) i karaktereve në thonjëza (") quhet **string**, string i karaktereve, mesazh apo **string literal** (vlerë e fiksuar string)
- Klasa **Console** i lejon aplikacionet C# të lexojnë dhe të shfaqin karaktere në dritaren e konsolës (dritaren komanduese).
- Metoda **Console.WriteLine** e shfaq argumentin e saj në dritaren e konsolës, të pasuar nga një karakter **newline** (dalje në rresht të ri) për ta vendosur (pozicionuar) kursoren e ekranit në fillimin e rreshtit tjetër.
- Shumica e urdhrrave (statements) përfundojnë me pikëpresje.

3.3 Krijimi i një aplikacioni të thjeshtë në Visual Studio

3.3.1 Krijimi i *Console App* (aplikacionit për konsolë)

- Visual Studio ofron shumë mënyra për ta personalizuar përjetimin tuaj të kodimit.
- Mund t'i ndryshosh settings (rregullimet) e editorit për t'i shfaqur numrat e rreshtave apo për ta caktuar **code indentation** (nivelizimin e kodit).

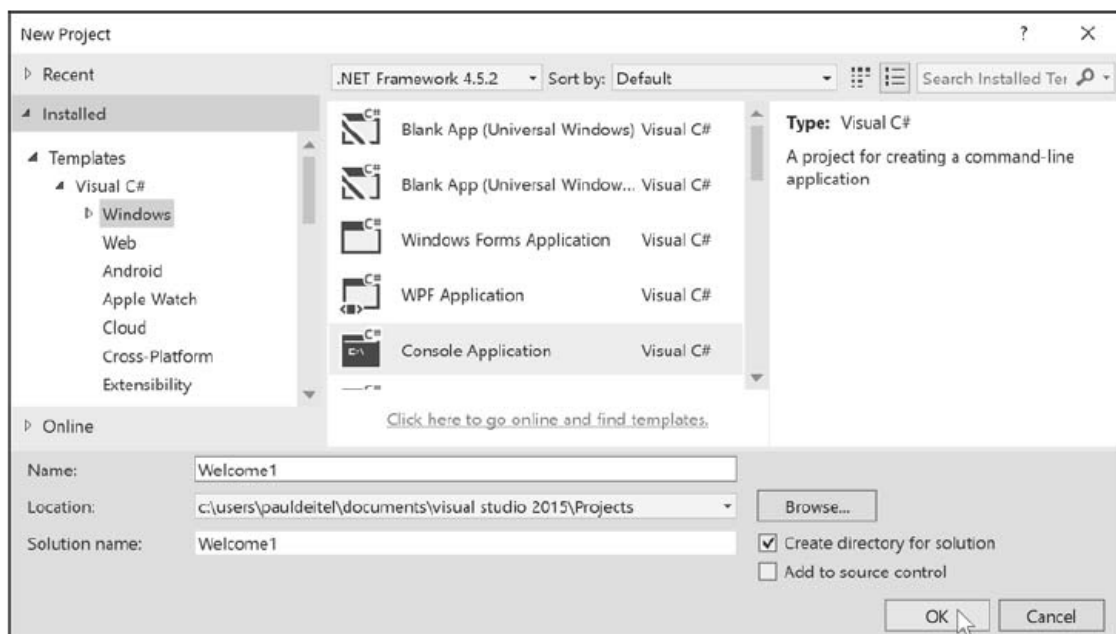


Fig. 3.3 | Selecting Console Application in the New Project dialog.

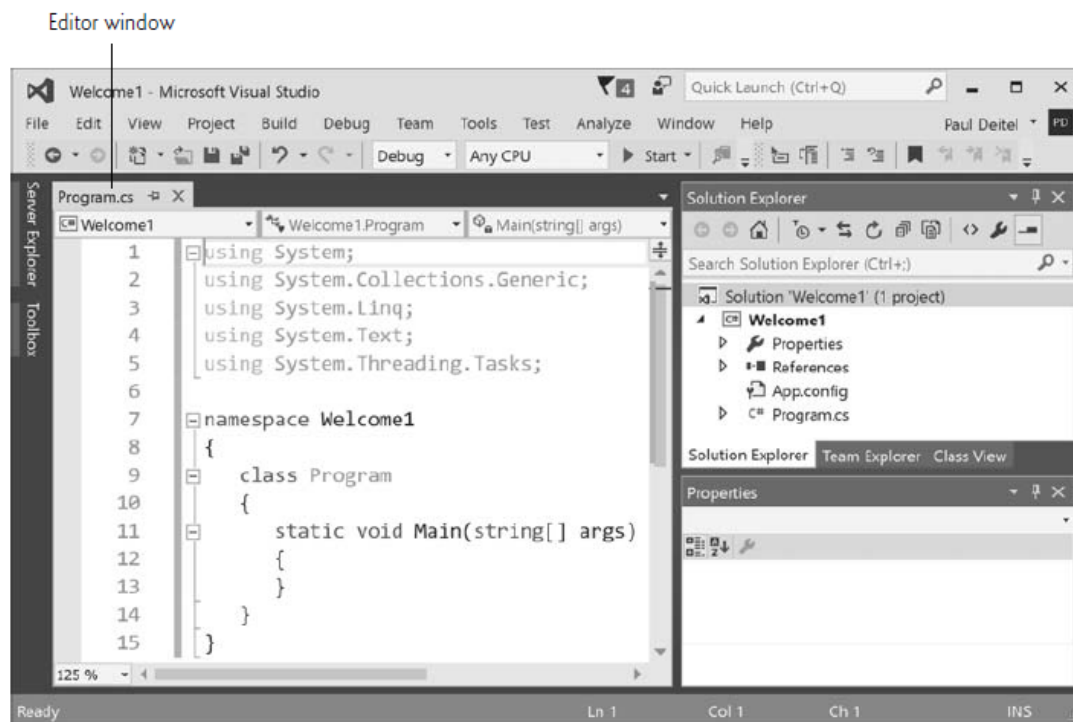


Fig. 3.4 | IDE with an open console app's code displayed in the editor and the project's contents shown in the **Solution Explorer** at the IDE's top-right side.

3.3.3 Shkrimi i kodit dhe përdorja e *IntelliSense*

- Gjersa i shkruani karakteret, në disa rrethana (kontekste) Visual Studio e thekson anëtarin e parë që përputhet me të gjithë karakteret e shkruara, pastaj e shfaq një tooltip që e përmban përshkrimin e atij anëtari. Kjo veçori e IDE-së quhet *IntelliSense*.
- Kur e shkruani kodin, IDE ose e aplikon **theksimin me ngjyra të sintaksës (syntax-color highlighting)** ose e gjeneron një **gabim sintaksor (syntax error)**.

a) *IntelliSense* window displayed as you type

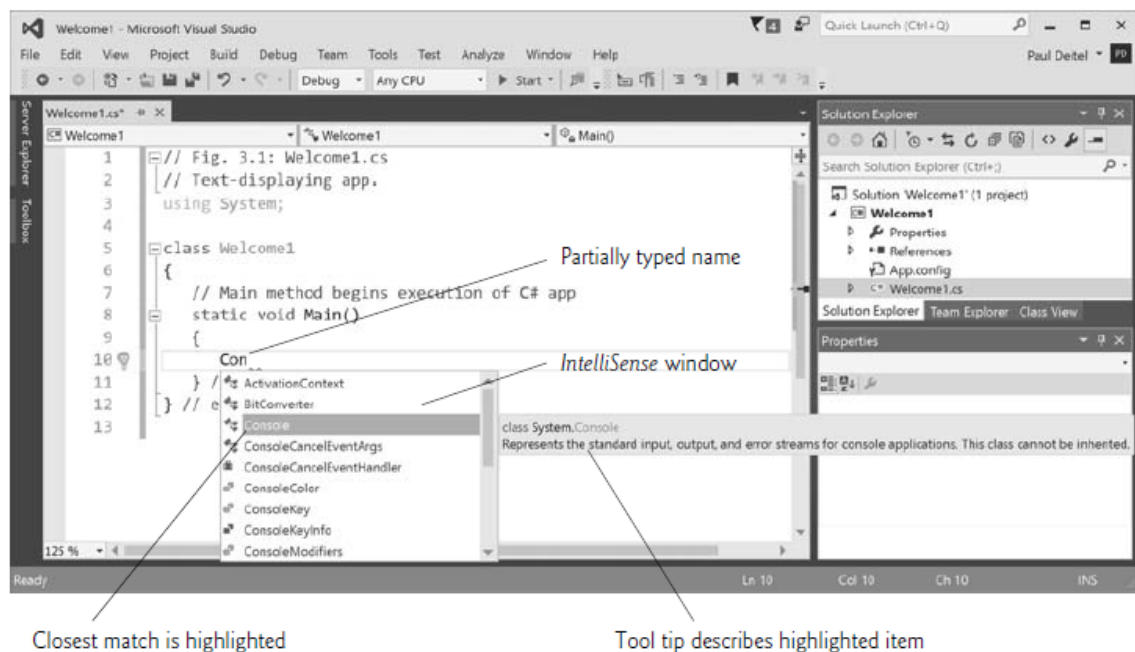


Fig. 3.5 | *IntelliSense* window as you type "Con".

b) *IntelliSense* window showing method names that start with *Write*

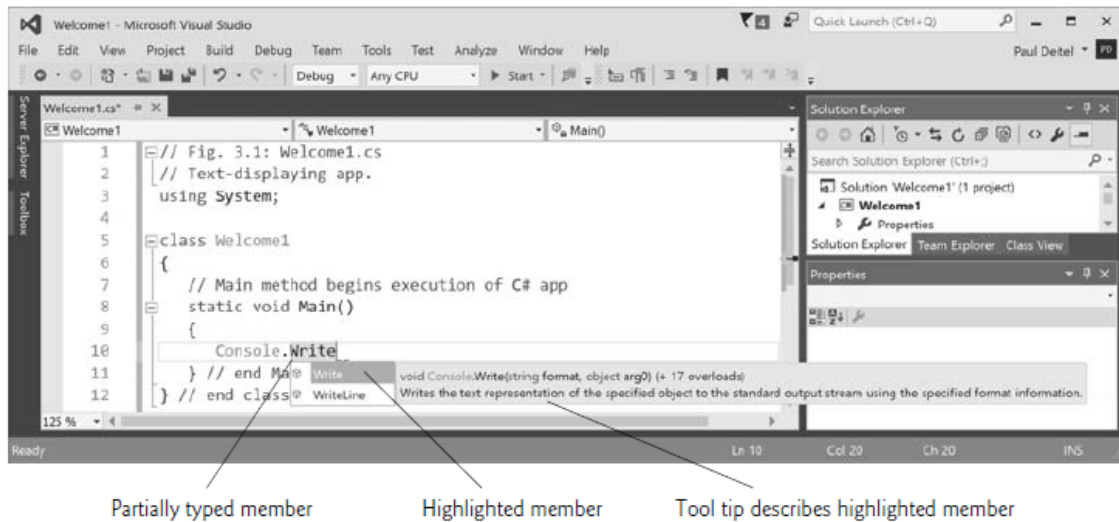


Fig. 3.6 | *IntelliSense* window.

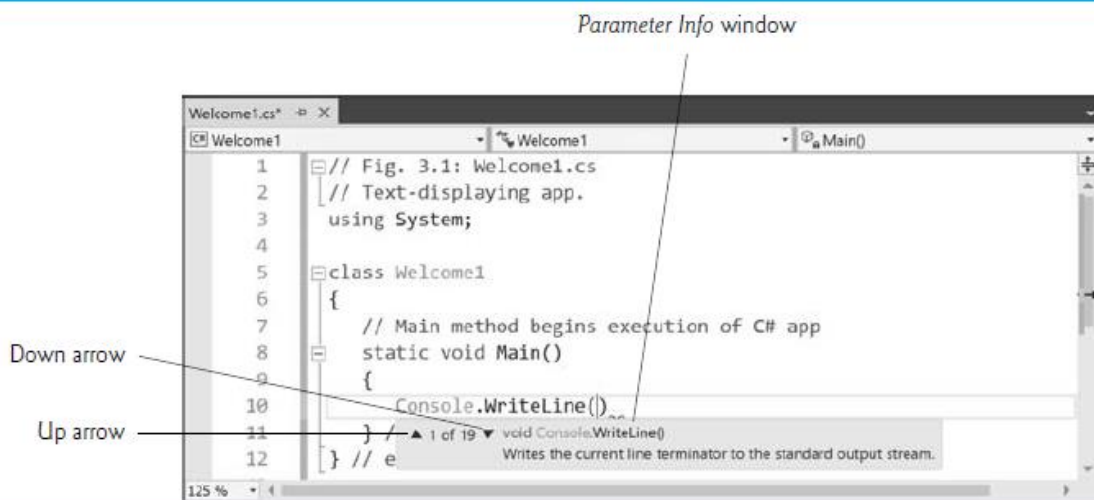


Fig. 3.7 | *Parameter Info* window.

3.3.5 Gabimet, mesazhet e gabimeve dhe Error List Window (Dritarja Error List, Lista e gabimeve)

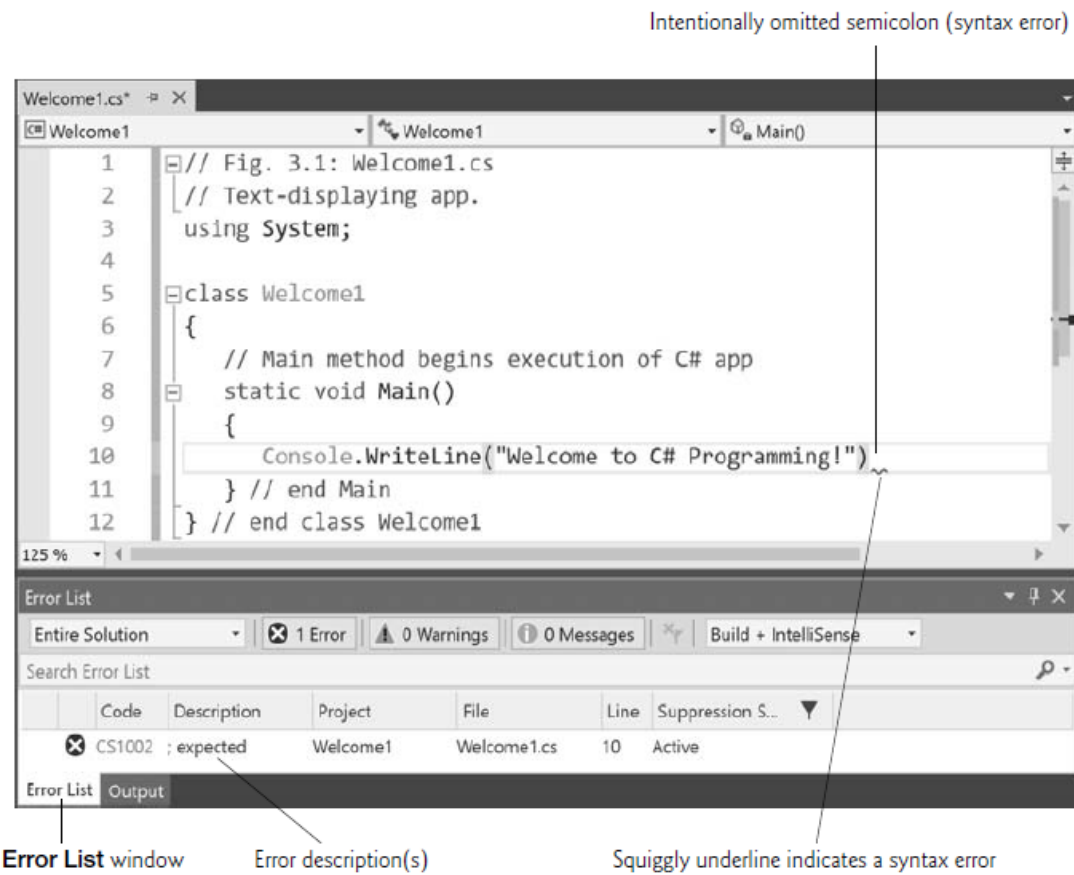


Fig. 3.9 | Syntax error indicated by the IDE.

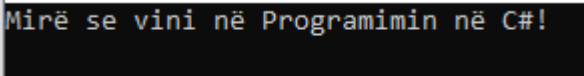
3.4 Ndryshimi (modifikimi) i aplikacionit tuaj të thjeshtë në C#

3.4.1 Shfaqja e një rreshti të tekstit me shumë urdhra

- **Console.Write** e shfaq argumentin e saj dhe e vendos (e pozicionon) kursorin e ekranit menjëherë pas karakterit të fundit të shfaqur.

```
// Fig. 3.10: MireSeVini2.cs
// Shfaqja e një rreshti të tekstit me shumë urdhra (statements, formulime).
using System;

class MireSeVini2
{
    // Metoda Main e nis ekzekutimin e aplikacionit në C#
    static void Main()
    {
        Console.Write("Mirë se vini në ");
        Console.WriteLine("Programimin në C#!");
        Console.ReadKey();
    } // përfundojë Main-in
} // përfundojë klasën MireSeVini2
```



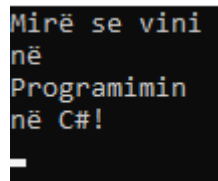
```
Mirë se vini në Programimin në C#!
```

3.4.2 Paraqitja e shumë rreshtave të tekstit me një urdhër të vetëm

- C# e kombinon një backslash (\ - thyesë mbrapsht) në një string me karakterin pasues në string për ta formuar një **escape sequence** (sekuencë të “ikjes”). Escape sequence **\n** (newline – rresht i ri) e vendos kursorin në fillimin e rreshtit tjetër.

```
// Fig. 3.11: MireSeVini3.cs
// Shfaqja e shumë rreshtave me një urdhër të vetëm.
using System;

class MireSeVini3
{
    // Metoda Main e nis ekzekutimin e aplikacionit në C#
    static void Main()
    {
        Console.WriteLine("Mirë se vini\n\nProgramimin\n\nC#!");
        Console.ReadKey();
    } // përfundojë Main-in
} // përfundojë klasën MireSeVini3
```



```
Mirë se vini
në
Programimin
në C#!
```

Sekuenca e ikjes (shmangies, arratisjes) (Escape sequence)	Përshkrimi
\n	Rresht i ri. E pozicionon kursorin e ekranit në fillim të rreshtit tjetër.
\t	Tab horizontal. E lëviz kursorin e ekranit në stacionin tjetër të tab-it.
\"	Thonjës (e dyfishtë). Përdoret për ta vendosur një karakter thonjës të dyfishtë (") në një string – p.sh. <code>Console.WriteLine("\"në thonjëza\"");</code> e shfaq "në thonjëza".
\r	Kthim në fillim (carriage return). E pozicionon kursorin e ekranit në fillim të rreshtit aktual – nuk e çon kursorin në rreshtin tjetër. Cilëndo karakterë të shkruar pas carriage return i mbishkruajnë karakterët e shkruar më herët në atë rresht.
\\	Backslash (thyesë mbrapsht). Përdoret për ta vendosur një karakter backslash në një string.

Fig. 3.12 | Sekuencat e zakonshme të ikjes.

3.5 Interpolimi (ndërfutja) e stringut

- **Urdhri për deklarim të ndryshores** (variable declaration statement) e specifikon emrin dhe tipin e variablës (ndryshore).
- **Variabla (ndryshoreja)** është lokacion (vend) në memorien (kujtesën) e kompjuterit ku mund të vendoset një vlerë për t'u përdorur më vonë në një aplikacion. Ndryshoret duhet të deklarohen me emër dhe tip para se të përdoren.
- Emri i ndryshores ia mundëson aplikacionit t'i qaset (ta aksesojë, access) vlerës së ndryshores në memorie. Emër i ndryshores mund të jetë çfarëdo identifikuesi valid.
- Sikur urdhrat e tjerë, urdhrat e deklarimit të ndryshoreve mbarojnë me **pikëpresje (;)**.
- Interpolimi i stringut (string interpolation) në C# 6 ju mundëson të futni vlera në një **string literal** (vlerë të fiksuar string).
- Një string i interpoluar duhet të fillojë me një **\$** (shenjë e dollarit). Pastaj mund t'i futni shprehjet e interpolimit të futura në kllapa gjarpërore, { }, kudo midis thonjzave ("") të stringut të interpoluar.
- Kur C# e has një varg të interpoluar, e zëvendëson çdo shprehje të kllapëzuar të interpolimit me vlerën përkatëse.

```
// Fig. 3.13: MireSeVini4.cs
// Fut përmbajtje në një string me interpolim (ndërfutje) të stringut.
using System;

class MireSeVini4
{
    // Metoda Main e nis ekzekutimin e aplikacionit në C#
    static void Main()
    {
        string personi = "Paul"; // variabël (ndryshore) që e ruan stringun "Paul"
        Console.WriteLine($"Mirë se vjen në programimin në C#, {personi}!");
        Console.ReadKey();
    } // përfundojë Main-in
} // përfundojë klasën MireSeVini4
```

```
Mirë se vjen në programimin në C#, Paul!
```

3.6 Një aplikacion tjetër në C#: Mbledhja e numrave të plotë

```
// Fig. 3.14: Mbledhja.cs
// Shfaqja e shumës së dy numrave të futur nga tastiera.
using System;

class Mbledhja
{
    // Metoda Main e nis ekzekutimin e aplikacionit në C#
    static void Main()
    {
        int numri1; // deklarohet numrin e parë që do të mbledhet
        int numri2; // deklarohet numrin e dytë që do të mbledhet
        int shuma; // deklarohet shumën e numrit1 dhe numrit2

        Console.WriteLine("Shkruaje numrin e parë të plotë: "); // nxite përdoruesin

        numri1 = int.Parse(Console.ReadLine()); // lexohet numrin e parë nga përdoruesi

        Console.WriteLine("Shkruaje numrin e dytë të plotë: "); // nxite përdoruesin

        numri2 = int.Parse(Console.ReadLine()); // lexohet numrin e dytë nga përdoruesi

        shuma = numri1 + numri2; // mbledhi numrat

        Console.WriteLine($"Shuma është {shuma}"); // shfaq shumën
        Console.ReadKey();
    } // përfundohet Main-in
} // përfundohet klasën Mbledhja
```

```
Shkruaje numrin e parë të plotë: 5
Shkruaje numrin e dytë të plotë: 7
Shuma është 12
```

3.6.1 Deklarimi i ndryshores së tipit *int*

- Integer-at janë numra të plotë, si -22, 7, 0 dhe 1024.
- Tipi **int** përdoret për t'i deklaruar ndryshoret që do të mbajnë vlera të plota. Rangu i vlerave për një **int** është prej -2,147,483,648 deri në +2,147,483,647.
- Tipet **float**, **double** dhe **decimal** specifikojnë numra realë, dhe tipi **char** specifikon shënime karakter. Numrat realë janë numrat që mund të përmbajnë pikë dhjetore, siç janë 3.4, 0.0 dhe -11.19. Ndryshoret e tipit të shënimeve **char** paraqesin karaktere individuale, si një shkronjë e madhe (p.sh. A), një shifër (p.sh. 7), një karakter special (p.sh. * apo %) apo një escape sequence (p.sh. karakteri për rresht të ri, \n).
- Tipet si **int**, **float**, **double**, **decimal** dhe **char** shpesh quhen tipe të thjeshta. Emrat e tipeve të thjeshta janë fjalë kyçe (keywords) prandaj duhet të shfaqen me të gjitha shkronjat të vogla.

3.6.3 Kërkimi i të dhënave hyrëse nga përdoruesi

- **Prompt-i** (inkurajimi, p.sh. kërkimi i të dhënave hyrëse) e udhëzon përdoruesin ta ndërmarrë një veprim specifik.
- Metoda **ReadLine** e klasës **Console** e merr një rresht të tekstit nga tastiera për përdorim në aplikacion.
- Metoda **Parse** e **int**-it e nxjerr një numër të plotë nga një string i karaktereve.
- Operatori i caktimit të vlerës (ndarjes së vlerës, dhënies së vlerës, assignment), =, ia mundëson aplikacionit t'ia japë një vlerë ndryshore. Operatori = quhet operator binar sepse i ka dy operandë. Urdhri i caktimit të vlerës e përdor operatorin e caktimit të vlerës për t'ia caktuar vlerën ndryshore.

3.6.6 Mbledhja e dy numrave

- Pjesët e urdhrave që kanë vlera quhen **shprehje** (**expressions**).

3.7 Koncepte të memories

- Emrat e ndryshoreve u korrespondojnë (u përgjigjen) lokacioneve (vendndodhjeve) në memorien e kompjuterit. Çdo ndryshore ka **emër**, **tip**, **madhësi** dhe **vlerë**.
- Kurdo që një vlerë vendoset në një lokacion në memorie, vlera e zëvendëson vlerën e mëparshme në atë lokacion. Vlera e mëparshme humbet.

number1	45
---------	----

Fig. 3.15 | Memory location showing the name and value of variable number1.

number1	45
number2	72

Fig. 3.16 | Memory locations after storing values for number1 and number2.

number1	45
number2	72
sum	117

Fig. 3.17 | Memory locations after calculating and storing the sum of number1 and number2.

3.8 Aritmetika

- Shumica e aplikacioneve bëjnë llogaritje aritmetike. Operatorët aritmetikë janë + (mbledhja), - (zbritja), * (shumëzimi), / (pjesëtimi) dhe % (mbetja gjatë pjesëtimit).
- Nëse të dy operandët e operatorit të pjesëtimit (/) janë numra të plotë (integers), rezultati është numër i plotë (integer).
- Operatori i mbetjes gjatë pjesëtimit, %, e jep pjesën e mbetur pas pjesëtimit.
- Shprehjet aritmetike në C# duhet të shkruhen në formë drejtvizore.
- Nëse një shprehje përmban kllapa të ndërfutura (nested parentheses), së pari llogaritet vlera në kllapat më të brendshme.
-

Operacioni në C#	Operatori aritmetik	Shprehja algjebrike	Shprehja në C#
Mbledhja	+	$f + 7$	<code>f + 7</code>
Zbritja	-	$p - c$	<code>p - c</code>
Shumëzimi	*	$b \cdot m$	<code>b * m</code>
Pjesëtimi	/	x / y apo $\frac{x}{y}$ apo $x \div y$	<code>x / y</code>
Mbetja	%	$r \bmod s$	<code>r % s</code>

Fig. 3.18 | Operatorët aritmetikë.

3.8.3 Rregullat e përparësisë së operatorëve (operator precedence)

- C# i aplikon operatorët në shprehjet aritmetike në një sekuencë (renditje) të saktë të përcaktuar nga rregullat e përparësisë së operatorëve.
- Asociativiteti (shoqërimi) e përcakton se a aplikohen operatorët nga e majta në të djathtë apo nga e djathta në të majtë.
- Kllapat redundante (të shtuara me qëllim) në një shprehje mund ta bëjnë një shprehje më të qartë.

Operatorët	Operacionet (veprimet)	Radha e evaluimit (llogaritjes, vlerësimit) (asociativiteti, shoqërueshmëria)
Evaluohen të parat		
*	Shumëzimi	Nëse ka disa operatorë të këtij tipi, ata evaluohen nga e majta në të djathtë
/	Pjesëtimi	
%	Mbetja	
Evaluohen më pas		
+	Mbledhja	Nëse ka disa operatorë të këtij tipi, ata evaluohen nga e majta në të djathtë
-	Zbritja	

Fig. 3.19 | Përparësia e operatorëve aritmetikë.

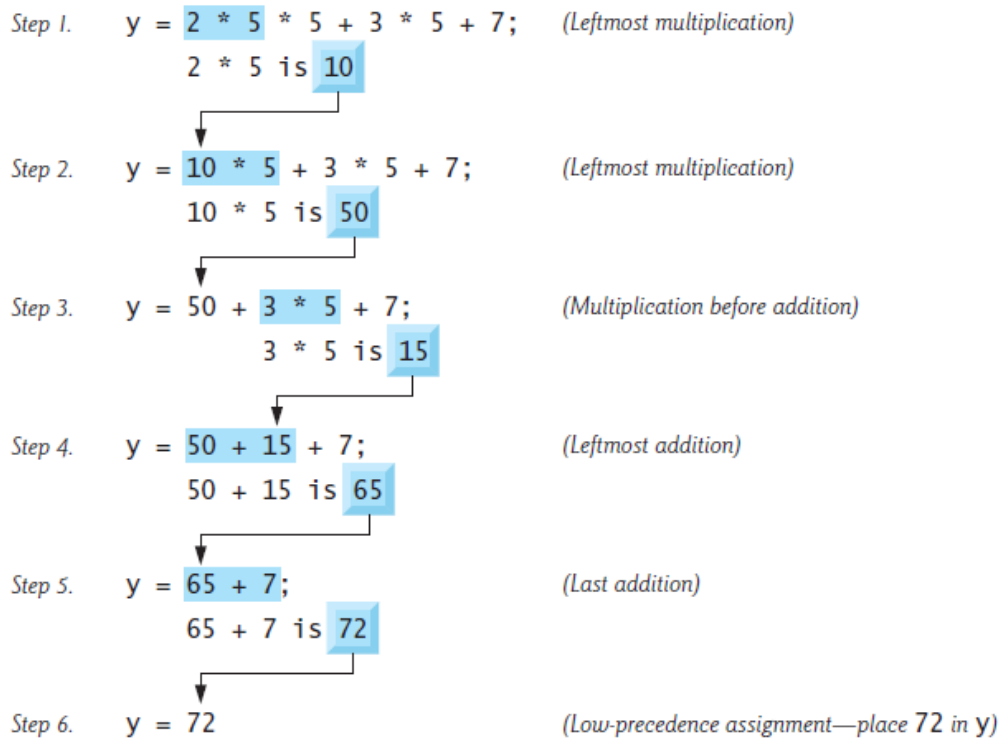


Fig. 3.20 | Order in which a second-degree polynomial is evaluated.

3.9 Marrja e vendimeve: Barazia dhe Operatorët Relacionalë

- Një kusht (condition) është një shprehje që mund të jetë ose e vërtetë (**true**) ose e pavërtetë (**false**). Urdhri **if** i C# ia lejon një aplikacioni ta marrë një vendim duke u bazuar në vlerën e një kushti.
- Kushtet në urdhrat **if** mund të formohen duke i përdorur operatorët e barazisë (`==` dhe `!=`) dhe operatorët relacionalë (`>`, `<`, `>=`, `<=`).
- Një urdhër **if** gjithmonë fillon me fjalën kyçe **if**, të pasuar nga një kusht në kllapa të vogla: `()`, dhe një trup, të cilin gjithmonë duhet ta vendosni në kllapa gjarpërore: `{ }`
- Urdhri i zbrazët (empty statement) është urdhër që nuk kryen ndonjë detyrë.

Operatorët standardë algjebrikë të barazisë dhe relacionale	Operatori i barazisë apo relacion në C#	Shembull i kushtit në C#	Kuptimi i kushtit në C#
Operatorët relacionale			
>	>	$x > y$	x është më i madh se y
<	<	$x < y$	x është më i vogël se y
\geq	>=	$x \geq y$	x është më i madh ose i barabartë me y
\leq	<=	$x \leq y$	x është më i vogël ose i barabartë me y
Operatorët e barazisë			
=	==	$x == y$	x është i barabartë me y
≠	!=	$x != y$	x nuk është i barabartë me y

Fig. 3.21 | Operatorët relacionale dhe të barazisë.

```
// Fig. 3.22: Krahاسimi.cs
// Krahاسimi i numrave të plotë duke përdorur urdhra if,
// operatorë të barazisë, dhe operatorë relacionale.
using System;

class Krahاسimi
{
    // Metoda Main e nis ekzekutimin e aplikacionit në C#
    static void Main()
    {
        // nxite përdoruesin dhe lexoje numrin e parë
        Console.WriteLine("Shkruaje numrin e parë të plotë: ");
        int numri1 = int.Parse(Console.ReadLine());

        // nxite përdoruesin dhe lexoje numrin e dytë
        Console.WriteLine("Shkruaje numrin e dytë të plotë: ");
        int numri2 = int.Parse(Console.ReadLine());

        if (numri1 == numri2)
        {
            Console.WriteLine($"{numri1} == {numri2}");
        }

        if (numri1 != numri2)
        {
            Console.WriteLine($"{numri1} != {numri2}");
        }

        if (numri1 < numri2)
        {
            Console.WriteLine($"{numri1} < {numri2}");
        }

        if (numri1 > numri2)
        {
            Console.WriteLine($"{numri1} > {numri2}");
        }

        if (numri1 <= numri2)
        {
            Console.WriteLine($"{numri1} <= {numri2}");
        }

        if (numri1 >= numri2)
        {
            Console.WriteLine($"{numri1} >= {numri2}");
        }

        Console.ReadKey();
    } // përfundojë Main-in
} // përfundojë klasën Krahاسimi
```

```
Shkruaje numrin e parë të plotë: 12
Shkruaje numrin e dytë të plotë: 12
12 == 12
12 <= 12
12 >= 12
```

```
Shkruaje numrin e parë të plotë: 5
Shkruaje numrin e dytë të plotë: 7
5 != 7
5 < 7
5 <= 7
```

```
Shkruaje numrin e parë të plotë: 5000
Shkruaje numrin e dytë të plotë: 3000
5000 != 3000
5000 > 3000
5000 >= 3000
```

Operatorët	Asociativiteti (Shoqërueshmëria)	Tipi
* / %	nga e majta në të djathtë	shumëzues
+ -	nga e majta në të djathtë	mbledhës
< <= > >=	nga e majta në të djathtë	relacional
== !=	nga e majta në të djathtë	barazi
=	nga e djathta në të majtë	caktim i (ndarje e) vlerës

Fig. 3.23 | Përparësia dhe asociativiteti i (shoqërueshmëria e) operatorëve të diskutuar deri tash.

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();

        // shfaqe 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.WriteLine("Shkruaje emrin: "); // kërkesa apo inkurajimi
        string emri = Console.ReadLine(); // lexoje emrin
        llogariaIme.CaktojeEmrin(emri); // vendose emrin në objektin llogariaIme

        // shfaqe 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 emriILlogarise)
    {
        emri = emriILlogarise; // 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 deklaram i klasës e përmban fjalën kyçe **class** që pasohet menjëherë nga emri i klasës.
- Çdo deklaram 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 parametrarit. 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 i kryejnë detyrat e metodës.
- Variablat e deklaruara 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**.
- Metodatat (dhe anëtarët e tjerë të klasës) që janë të deklaruara **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.

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

- Diagramet e klasave në UML i përmbledhin atributet dhe operacionet e klasës.
- Diagramet UML u ndihmojnë dizajnuesve (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.

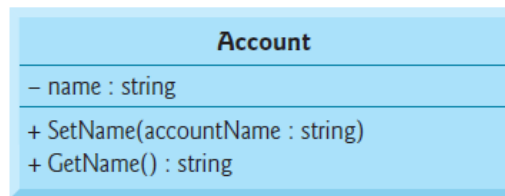


Fig. 4.3 | UML class diagram for class `Account` of Fig. 4.2.

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.WriteLine("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 aksesori (qasës) **set** për ta ruajtur një vlerë në një variabël dhe një aksesori **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 aksesoriin 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 aksesoriin (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

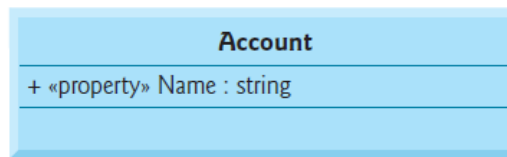


Fig. 4.7 | UML class diagram for class Account of Fig. 4.6.

- Properti-t e C# modelohen në UML si attribute. 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 stereotipe 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 aksesori (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**.

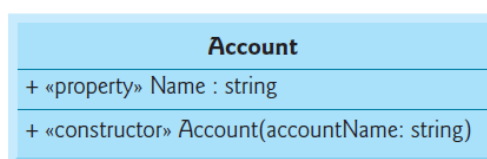


Fig. 4.10 | UML class diagram for `Account` class of Fig. 4.8.

- 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).
- Aksessori **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.WriteLine("\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.WriteLine("\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
```

Specifikuesi i formatit (Format specifier)	Përshkrimi
C apo c	E formaton string-un si valutë. E përfshin një simbolin të përshtatshëm të monedhës (\$) në SHBA) pas numrit. I ndan shifrat me një <i>karakter ndarës</i> të përshtatshëm (në SHBA është një presje midis çdo tri shifrave për mijëra, miliona, etj.) dhe e cakton numrin e shifrave decimale të jetë dy si e paracaktuar (default).
D apo d	E formaton string-un si numër të plotë (vetëm tipet e numrave të plotë).
N apo n	E formaton string-un me një ndarës të mijësheve dhe me dy shifra decimale si të paracaktuar (default).
E apo e	E formaton numrin duke e përdorur shënimin shkencor (scientific notation) me gjashtë shifra decimale (paracaktuar).
F apo f	E formaton string-un me një numër fiks të shifrave decimale (paracaktuar në dy).
G apo g	E formaton numrin normalisht me shifra decimale apo duke e përdorur shënimin shkencor, varësisht nga konteksti. Nëse një element i formatit nuk përmban specifikues të formatit, formati G supozohet në mënyrë implicite.
X apo x	E formaton string-un si heksadecimal (numrat me bazë 16).

Fig. 4.13 | Specifikuesit e formatit të string-ut

- 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 shkurtësë 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.

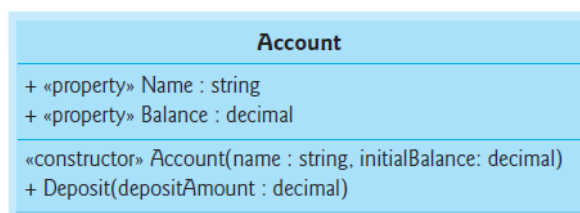


Fig. 4.14 | UML class diagram for **Account** class of Fig. 4.11.

5. Zhvillimi i algoritmeve dhe Urdhrat e kontrollit: Pjesa 1

5.2 Algoritmet

- Algoritmi është procedurë për zgjidhjen e një problemi në terma të veprimeve që duhet të ekzekutohen dhe të radhës në të cilën ekzekutohen këto veprime.
- Specifikimi i radhës në të cilën ekzekutohen urdhrat (veprimet, aksionet) në aplikacion quhet kontroll i programit.

5.3 Pseudokodi

- Pseudokodi është gjuhë joformale që ju ndihmon të zhvilloni algoritme pa pasur nevojë të shqetësoheni për detajet strikte të sintaksës së gjuhës C#.
- Pseudokodi i përgatitur me kujdes mund të konvertohet (shndërrohet) lehtë në aplikacion përkatës në C#.

```
1  Nxite përdoruesin ta shkruajë numrin e parë të plotë
2  Lexoje si hyrje numrin e parë të plotë
3
4  Nxite përdoruesin ta shkruajë numrin e dytë të plotë
5  Lexoje si hyrje numrin e dytë të plotë
6
7  Mbledhe numrin e parë dhe numrin e dytë të plotë, ruaje rezultatin
8  Shfaq rezultatin
```

Fig. 5.1 | Pseudokodi për programin për mbledhje në Fig. 3.14.

5.4 Strukturat e kontrollit

- Janë tri tipe (lloje) të strukturave të kontrollit:
 - sekuenca (sequence),
 - përzgjedhja (selection) dhe
 - përsëritja (iterimi, iteration).
- **Struktura sekuençë** është e integruar në C#. Nëse nuk udhëzohet ndryshe, kompjuteri i ekzekuton urdhrat e C# njërin pas tjetrit në radhën që janë shkruar.
- **Diagramet e aktivitetit** janë pjesë e UML. Një diagram i aktivitetit e modelon rrjedhën e punës së një pjese të sistemit softuerik.
- Diagramet e aktivitetit përbëhen nga simbole për qëllime të veçanta, si simbolet e gjendjes së veprimit, diamantet dhe rrathët e vegjël. Këto simbole janë të lidhura me shigjeta të tranzicionit (kalimit, ndryshimit të gjendjes), që e përfaqësojnë rrjedhën e aktivitetit.
- Si pseudokodi, diagramet e aktivitetit ju ndihmojnë të zhvilloni dhe të paraqitni algoritme. Diagramet e aktivitetit tregojnë qartë se si funksionojnë strukturat e kontrollit.
- Simbolet e gjendjes së veprimit (action-state symbols – drejtkëndëshat me qoshet të zëvendësuara me harqe) paraqesin veprime që duhet të kryhen.
- Shigjetat në diagram të aktivitetit paraqesin tranzicione, që e tregojnë radhën në të cilën ndodhin veprimet e paraqitura me gjendjet e veprimit.
- Rrethi i mbushur në diagram të aktivitetit e paraqet gjendjen fillestare të aktivitetit. Rrethi i mbushur i rrethuar nga një rreth i zbrazët e paraqet gjendjen përfundimtare.
- Drejtkëndëshat me qoshet e sipërme të djathta të palosura janë shënime UML (si komentet në C#) – vërejtje shpjeguese që e përshkruajnë qëllimin e simboleve në diagram.

5.4.1 Struktura sekuencë

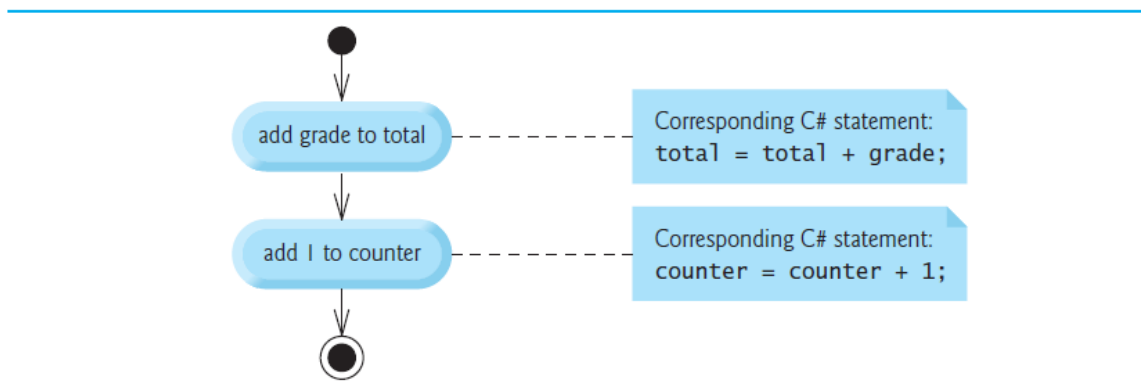


Fig. 5.2 | Sequence structure activity diagram.

5.4.2 Urdhrat e përzgjedhjes

- C# i ka tri tipe (lloje) të urdhrave të përzgjedhjes:
 - urdhrin **if**,
 - urdhrin **if...else** dhe
 - urdhrin **switch**.
- Urdhri **if** quhet urdhër **me një përzgjedhje** sepse e zgjedh apo e injoron vetëm një veprim (apo grup të veprimeve).
- Urdhri **if...else** quhet urdhër **me dy përzgjedhje** sepse zgjedh ndërmjet dy veprimeve (apo grupeve të veprimeve) të ndryshme.
- Urdhri **switch** quhet urdhër **me përzgjedhje të shumëfishtë** sepse zgjedh ndërmjet shumë veprimeve (apo grupeve të veprimeve) të ndryshme.

5.4.3 Urdhrat e përsëritjes (iterimit)

- C# i ofron katër urdhra të përsëritjes:
 - **while**,
 - **do...while**,
 - **for** dhe
 - **foreach**.
- Urdhrat **while**, **for** dhe **foreach** i kryejnë veprimet në trupat e tyre **zero apo më shumë herë**.
- Urdhri **do...while** i kryen veprimet në trupin e tij **një apo më shumë herë**.

5.4.4 Përmbledhje e urdhrave të kontrollit

- Urdhrat e kontrollit mund të lidhen në dy mënyra: **renditja** e urdhrave të kontrollit njëri pas tjetrit (**stacking**) apo **ndërfutja** e urdhrave të kontrollit njëri brenda tjetrit (**nesting**).

5.4.5 Urdhri *if* me një përzgjedhje

- Urdhri *if* me një përzgjedhje e kryen një veprim (apo grup të veprimeve) të caktuar vetëm kur kushti është i vërtetë; përndryshe veprimi anashkalohet.
- Në diagram të aktivitetit, simboli i diamantit tregon se duhet të merret një vendim. Rrjedha e punës do të vazhdojë përgjatë një rruge të përcaktuar nga kushtet vëzhguese (rojtare) që i ndërlidhen simbolit.
- Kur modelohen nga një diagram i aktivitetit UML, të gjithë urdhrat e kontrollit përmbajnë gjendje fillestare, shigjeta kalimtare (të tranzicionit), gjendje të veprimit dhe simbole të vendimit.

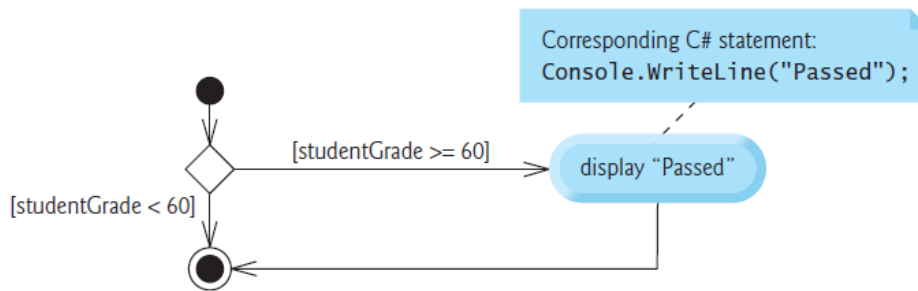


Fig. 5.3 | if single-selection statement UML activity diagram.

5.6 Urdhri *if...else* me përzgjedhje të dyfishtë

- Urdhri **if...else** ju lejon ta specifikoni një veprim (apo grup të veprimeve) për t'u kryer kur kushti është i vërtetë dhe një veprim (apo grup të veprimeve) tjetër kur kushti është i pavërtetë.
- Për t'i përfshirë disa urdhra në trupin e një **if**-i (apo trupin e një **else**-i për një urdhër **if...else**), futini urdhrat në kllapa gjarpërore (pra { dhe }).
- Grupi i urdhrave që përmbahet brenda një çifti të kllapave gjarpërore quhet **bllok**. Blloku mund të vendoset kudo në aplikacion në vendet ku mund të vendoset një urdhër i vetëm.
- C# e ofron operatorin kushtëzues (kondicional) (**? :**) që mund të përdoret në vend të një urdhri **if...else**. Shprehja kushtëzuese vlerësohet si operandi i dytë nëse operandi i parë del **true**, dhe vlerësohet si operandi i tretë nëse operandi i parë del **false**.

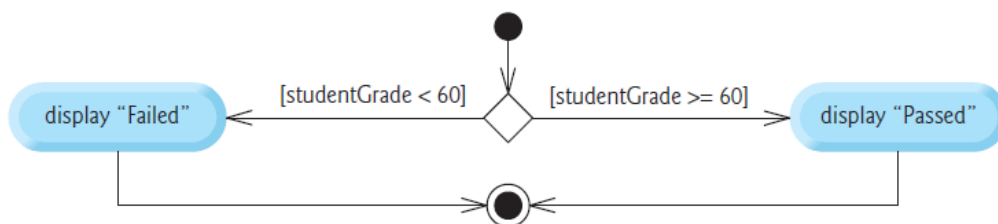


Fig. 5.4 | if...else double-selection statement UML activity diagram.

5.7 Klasa *Student*: Urdhrat e ndërfutur if...else

- Një properti (veti, property) që është read-only (vetëm për lexim) e përmban vetëm një aksesor (qasës) get.

```
// Fig. 5.5: Student.cs
// Klasa Student që e ruan emrin dhe mesataren e një studenti.
using System;

class Student
{
    public string Emri { get; set; } // properti (property, veti)
    private int mesatarja; // variabla (ndryshorja) e instancës

    // konstruktori i inicializon proprietit Emri dhe Mesatarja
    public Student(string emriIStudentit, int mesatarjaEStudentit)
    {
        Emri = emriIStudentit;
        Mesatarja = mesatarjaEStudentit; // e cakton variablën e instancës mesatarja
    }

    // properti për ta marrë (get) dhe caktuar (set)
    // variablën e instancës mesatarja
    public int Mesatarja
    {
        get // e kthen mesataren e Studentit
        {
            return mesatarja;
        }
        set // e cakton mesataren e Studentit
        {
            // validej që mesatarjaEStudentit është > 0 dhe <= 100; përndryshe,
            // mbaje vlerën aktuale të variablës së instancës mesatarja
            if (value > 0)
            {
                if (value <= 100)
                {
                    mesatarja = value; // caktoj vlerën variablës së instancës
                }
            }
        }
    }
}
```

```
// e kthen notën shkronjore të Studentit, sipas mesatares
public string NotaShkronjore
{
    get
    {
        string notaShkronjore = string.Empty; // string.Empty është ""

        if (mesatarja >= 90)
        {
            notaShkronjore = "A";
        }
        else if (mesatarja >= 80)
        {
            notaShkronjore = "B";
        }
        else if (mesatarja >= 70)
        {
            notaShkronjore = "C";
        }
        else if (mesatarja >= 60)
        {
            notaShkronjore = "D";
        }
        else
        {
            notaShkronjore = "F";
        }

        return notaShkronjore;
    }
}
}
```

```
// Fig. 5.6: TestIStudentit.cs
// Krijo dhe testo objekte Student-ë.
using System;

class TestIStudentit
{
    static void Main()
    {
        Student studentja1 = new Student("Jane Green", 93);
        Student studenti2 = new Student("John Blue", 72);

        Console.Write($"Barasvlera si notë shkronjore e {studentja1.Emri} për ");
        Console.WriteLine($"{studentja1.Mesatarja} është {studentja1.NotaShkronjore}");
        Console.Write($"Barasvlera si notë shkronjore e {studenti2.Emri} për ");
        Console.WriteLine($"{studenti2.Mesatarja} është {studenti2.NotaShkronjore}");
        Console.ReadKey();
    }
}
```

```
Barasvlera si notë shkronjore e Jane Green për 93 është A
Barasvlera si notë shkronjore e John Blue për 72 është C
```

5.8 Urdhri i përsëritjes (iterimit) *while*

- Një urdhër i përsëritjes ju lejon të specifikoni që një aplikacion duhet ta përsërisë një veprim përderisa njëfarë kushti të mbetet i vërtetë (true).

- Format i për urdhrin e iterimit **while** është

```
while (kushti)
{
    urdhri
}
```

- UML i paraqet si diamante edhe simbolin e bashkimit (merge) edhe simbolin e vendimit. Simboli i bashkimit i bashkon dy rrjedha të aktivitetit në një.

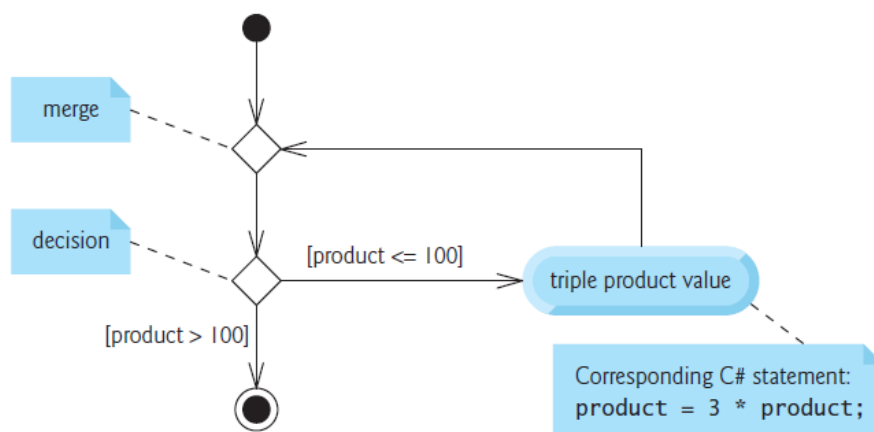


Fig. 5.7 | while iteration statement UML activity diagram.

5.9 Formulimi i algoritmeve: Përsëritja (iterimi) i kontrolluar me numërues (counter-controlled iteration)

- Përsëritja e (iterimi i) kontrolluar me numërues është teknikë që e përdor një variabël (ndryshore) të quajtur numërues (counter) për ta kontrolluar se sa herë do të ekzekutohet një grup i urdhrave.
- C# kërkon që patjetër ndryshoreve lokale t'u caktohet vlerë para se t'u përdoren vlerat.
- Pjesëtimi i dy numrave të plotë rezulton në pjesëtim të plotë – çfarëdo pjese thyesore e llogaritjes humbet.

```
1    Caktoje totalin të jetë zero
2    Caktoje numëruesin e notave të jetë një
3
4    Përderisa numëruesi i notave është më i vogël ose i barabartë me 10
5        Kërko nga përdoruesi ta shkruajë si hyrje notën tjetër
6        Merre si hyrje notën tjetër
7        Shtoja notën totalit
8        Shtoja një numëruesit të notave
9
10   Caktoje mesataren e klasës të jetë totali i pjesëtuar me 10
11   Shfaqe mesataren e klasës
```

Fig. 5.8 | Algoritmi në pseudokod që e përdor përsëritjen (iteracionin, iterimin) e kontrolluar me numërues (counter-controlled iteration) për ta zgjidhur problemin e mesatares së klasës

```
// Fig. 5.9: MesatarjaEKlases.cs
// Zgjidhja e problemit të mesatares së klasës duke e përdorur
// përsëritjen (iteracionin, iterimin) e kontrolluar me numërues
// (counter-controlled iteration)

using System;

class MesatarjaEKlases
{
    static void Main()
    {
        // faza e inicializimit
        int totali = 0; // inicializojë shumën e notave të shkruara nga përdoruesi
        int numeruesiINotave = 1; // inicializojë numrin rendor të notës
        // që do të futet si hyrje

        // faza e procesimit e përdor përsëritjen
        // (iteracionin, iterimin) e kontrolluar me numërues
        while (numeruesiINotave <= 10) // sillu 10 herë në cikël (unazë, loop)
        {
            Console.Write(
                $"Shkruaje notën #{numeruesiINotave}: "); //bëja kërkesën përdoruesit
            int nota = int.Parse(Console.ReadLine()); // merre notën si hyrje
            totali = totali + nota; // shtoja notën totalit
            numeruesiINotave = numeruesiINotave + 1; // rrite numëruesin për 1
        }

        // faza e përfundimit
        int mesatarja = totali / 10; // pjesëtimi i numrave të plotë (integer)
        // jep rezultat numër të plotë

        // shfaqje totalin dhe mesataren e notave
        Console.WriteLine($"\\nTotali i 10 notave është {totali}");
        Console.WriteLine($"Mesatarja e klasës është {mesatarja}");
    }
}
```

```
Shkruaje notën #1: 88
Shkruaje notën #2: 79
Shkruaje notën #3: 95
Shkruaje notën #4: 100
Shkruaje notën #5: 48
Shkruaje notën #6: 88
Shkruaje notën #7: 92
Shkruaje notën #8: 83
Shkruaje notën #9: 90
Shkruaje notën #10: 85

Totali i 10 notave është 848
Mesatarja e klasës është 84
```

5.10 Formulimi i algoritmeve: Përsëritja (iterimi) i kontrolluar me vlerë vëzhguese (sentinel-controlled iteration)

- Përsëritja e kontrolluar me vlerë vëzhguese (rojtare, dalëse) është teknikë që e përdor një vlerë të veçantë që quhet vlerë vëzhguese për ta treguar “përfundimin e futjes së shënimeve.”
- Përsëritja e kontrolluar me vlerë vëzhguese shpesh quhet **iterim indefinitiv** (përsëritje e pacaktuar), sepse numri i përsëritjeve nuk dihet paraprakisht.
- Detajimi gradual lart-poshtë, hap pas hapi (top-down, stepwise refinement) është esencial (thelbësor) për zhvillimin e aplikacioneve të strukturuar mirë.
- Filloni me një paraqitje të pseudokodit të majës (top) – një formulim i vetëm që, në fakt, është përfaqësim i plotë i një aplikacioni.
- Maja (top) rrallë përcjell (jep, përçon) detaje të mjaftueshme nga të cilat mund të shkruhet një aplikacion. Prandaj në detajimin e parë gradual, e detajojmë (e përmirësojmë, e përsosim) majën në seri të detyrave më të vogla dhe i rendisim ato sipas radhës në të cilën do të kryhen. Ky detajim përdor vetëm detyra në sekuencë (radhë).
- Në detajimin e dytë, përqendrohemi te ndryshoret dhe logjika specifike.
- Një numër me pikë dhjetore quhet numër real apo numër me pikë lëvizëse (p.sh. 7.33, 0.0975 apo 1000.12345).
- Tipet **float** dhe **double** i paraqesin numrat me pikë lëvizëse.
- Ndryshoret **double** zakonisht mund të ruajnë numra më të mëdhenj dhe me detaje më të imëta (d.m.th. me më shumë saktësi/precizitet).
- Operatori unar i shndërrimit (cast) të tipit (**double**) e krijon një kopje të *përkohshme* me pikë lëvizëse të operandit të tij. Përdorja e operatorit të shndërrimit në këtë mënyrë quhet konvertim (shndërrim) ekspllicit.
- Për t’u siguruar se të dy operandët e një operatori binar janë të tipit të njëjtë, C# bën promovim (ngritje të nivelit të tipit) në operandët e përzgjedhur.
- Specifikuesi i formatit **F** përdoret për t’i formatizuar numrat me pikë lëvizëse – zakonisht të rrumbullakësuar në dy shifra në të djathtë të pikës decimale.

- 1 *Inicializojë totalin të jetë zero*
- 2 *Inicializojë numëruesin të jetë zero*
- 3
- 4 *Kërko nga përdoruesi ta shkruajë si hyrje notën e parë*
- 5 *Merre si hyrje notën e parë (që mund të jetë vlera vëzhguese)*
- 6
- 7 *Përderisa përdoruesi nuk e ka shkruar ende vlerën vëzhguese*
- 8 *Shtoja këtë notë totalit aktual*
- 9 *Shtoja një numëruesit të notave*
- 10 *Kërko nga përdoruesi ta shkruajë si hyrje notën tjetër*
- 11 *Merre si hyrje notën tjetër (që mund të jetë vlera vëzhguese)*
- 12
- 13 *Nëse numëruesi nuk është baraz me zero*
- 14 *Caktoje mesataren të jetë totali i pjesëtuar me numëruesin*
- 15 *Shkruaje në dalje mesataren*
- 16 *Përndryshe*
- 17 *Shkruaje në dalje "Nuk u shkrua asnjë notë"*

Fig. 5.10 | Algoritmi në pseudokod i nxjerrjes së mesatares së klasës me përsëritje të kontrolluar me vlerë vëzhguese (sentinel-controlled iteration).


```

// Fig. 5.11: MesatarjaEKlases2.cs
// Zgjidhja e problemit të mesatares së klasës duke e përdorur
// përsëritjen (iteracionin, iterimin) e kontrolluar me vlerë vëzhguese
// (sentinel-controlled iteration).

using System;
class MesatarjaEKlases2
{
    static void Main()
    {
        // faza e inicializimit
        int totali = 0; // inicializojë shumën e notave
        int numeruesiINotave = 0; // inicializojë nr. e notave të regj. deri tash

        // faza e procesimit
        // bëja kërkesën përdoruesit dhe lexojë notën
        Console.WriteLine("Shkruaje notën #1 apo -1 për t'u ndalur: ");
        int nota = int.Parse(Console.ReadLine());

        // sillu në cikël deri kur të lexohet vlera vëzhguese nga përdoruesi
        while (nota != -1)
        {
            totali = totali + nota; // shtojë notën totalit
            numeruesiINotave = numeruesiINotave + 1; // rrite numëruesin

            // bëja kërkesën për shënimin hyrës dhe lexojë notën nga përdoruesi
            Console.WriteLine(
                $"Shkruaje notën #{numeruesiINotave + 1} apo -1 për t'u ndalur: ");
            nota = int.Parse(Console.ReadLine());
        }

        // faza e përfundimit
        // nëse përdoruesi e ka shkruar të paktën një notë...
        if (numeruesiINotave != 0)
        {
            // përdor numër me pikë decimale për ta llogaritur mesataren e notave
            double mesatarja = (double)totali / numeruesiINotave;

            // shfaqë totalin dhe mesataren (me saktësi prej dy shifrash)
            Console.WriteLine(
                $"Totali i {numeruesiINotave} notave të regjistruara është {totali}");
            Console.WriteLine($"Mesatarja e klasës është {mesatarja:F}");
        }
        else // nuk është regjistruar asnjë notë, prandaj shfaq mesazh të gabimit
        {
            Console.WriteLine("Nuk është regjistruar asnjë notë");
        }
    }
}

```

```

Shkruaje notën #1 apo -1 për t'u ndalur: 97
Shkruaje notën #2 apo -1 për t'u ndalur: 88
Shkruaje notën #3 apo -1 për t'u ndalur: 72
Shkruaje notën #4 apo -1 për t'u ndalur: -1

Totali i 3 notave të regjistruara është 257
Mesatarja e klasës është 85.67

```

5.11 Formulimi i algoritmeve: Urdhrat e kontrollit që janë të ndërfutur (nested)

Për shembullin vijues e formulojmë një algoritëm duke përdorur **pseudokod** dhe **detajimin gradual lart-poshtë, hap pas hapi (top-down, stepwise refinement)**, dhe e shkruajmë një aplikacion përkatës në C#.

Formulimi i problemit

Le ta shohim formulimin vijues të problemit:

Një kolegji e ofron një kurs që i përgatit studentët për provimin shtetëror për ndërmjetësues për pasuri të patundshme. Vitin e fundit, në provim hynë 10 nga studentët që e mbaruan këtë kurs. Kolegji dëshiron ta dijë se sa mirë dolën studentët e tij në provim. Ju kanë kërkuar ta shkruani një aplikacion për t'i përmbledhur rezultatet. Jua kanë dhënë listën e 10 studentëve. Pranë secilit emër është shkruar një 1sh nëse studenti e ka kaluar provimin ose një 2sh nëse studenti ka dështuar. Aplikacioni juaj duhet t'i analizojë rezultatet e provimit si vijon:

1. *Merre si hyrje çdo rezultat të testit (d.m.th. një 1sh apo një 2sh). Shfaqe mesazhin "Shkruaje rezultatin" në ekran secilën herë që aplikacioni e kërkon një rezultat të testit.*
2. *Numëroje sasinë e rezultateve të testit të secilit lloj.*
3. *Shfaqe një përmbledhje të rezultateve të testit, duke e treguar numrin e studentëve që kaluan dhe numrin e atyre që dështuan.*
4. *Nëse provimin e kaluan mbi 8 studentë, shfaqe mesazhin "Bonus për instruktorin!"*

Gjetjet që i vërehen nga formulimi i problemit

Pas leximit të formulimit të problemit, i vërehen gjetjet vijuese:

1. Aplikacioni duhet t'i procesojë rezultatet e testit për 10 studentë. Mund të përdoret një **cikël i kontrolluar nga numëruesi (counter-controlled loop)** sepse numri i rezultateve të testit dihet paraprakisht.
2. Secili rezultat i testit ka vlerë numerike – ose një 1sh ose një 2sh. Sa herë që aplikacioni e lexon një rezultat të testit duhet ta përcaktojë se a është numri 1 apo 2. E testojmë për 1 në algoritmin tonë. Nëse numri nuk është 1, supozojmë se është 2. (Ushtrimi 5.24 i shqyrton pasojat e këtij supozimi.)
3. I përdorim dy numërues për t'i përcjellë rezultatet e provimit – një për ta numëruar numrin e studentëve që e kaluan provimin dhe një për ta numëruar numrin e studentëve që dështuan në provim.
4. Pasi t'i ketë procesuar të gjitha rezultatet, aplikacioni duhet ta përcaktojë se a e kaluan provimin mbi tërë studentët.

Detajimi gradual lart-poshtë, hap pas hapi (top-down, stepwise refinement): Paraqitja në pseudokod e majës (top)

Le të vazhdojmë me detajimin gradual lart-poshtë, hap pas hapi (top-down, stepwise refinement). Fillojmë me paraqitjen në pseudokod të majës:

Analizojë rezultatet e provimit dhe vendos se a duhet të marrë bonus instruktori

Maja (top) është paraqitja e plotë e aplikacionit, por me gjasë do të duhen disa detajime para se pseudokodi të mund të evoluojë natyrshëm në aplikacion në C#.

Detajimi gradual lart-poshtë, hap pas hapi (top-down, stepwise refinement): Detajimi i parë

Detajimi ynë i parë është:

Inicializoji variablat

Merri si hyrje 10 rezultatet e provimit dhe numëroji kalimet dhe dështimet

Shfaqe përmbledhjen e rezultateve të provimit dhe vendos a duhet të marrë bonus instruktori

Edhe këtu, megjithëse e kemi një paraqitje të plotë të tërë aplikacionit, nevojitet detajim i mëtejshëm. Tash i specifikojmë ndryshoret individuale. Duhet numëruar për t'i regjistruar kalimet dhe dështimet, një numërues do të përdoret për ta kontrolluar procesin e ciklimit dhe duhet një variabël për ta ruajtur shënimin hyrës të përdoruesit. Variabla në të cilën do të ruhet shënimi hyrës i përdoruesit nuk inicializohet në fillim të algoritmit, meqë vlera e saj lexohet nga përdoruesi gjatë secilës përsëritje (iterim) të ciklit.

Detajimi gradual lart-poshtë, hap pas hapi (top-down, stepwise refinement): Detajimi i dytë

Urdhri në pseudokod

Inicializoji variablat

mund të detajohet si vijon:

Inicializoji kalimet në zero

Inicializoji dështimet në zero

Inicializojë numëruesin e studentëve në një

Vëreni se vetëm numëruesit inicializohen në fillim të algoritmit.

Urdhri në pseudokod

Merri si hyrje 10 rezultatet e provimit dhe numëroji kalimet dhe dështimet

e kërkon një cikël që e merr si hyrje rezultatin e secilit provim. E dimë paraprakisht se janë saktësisht 10 rezultate të provimit, prandaj këtu është i përshtatshëm cikli i kontrolluar me numërues. Brenda ciklit (d.m.th. **i ndërfutur** brenda ciklit), një urdhër me përzgjedhje të dyfishtë do ta përcaktojë se a është secili rezultat i provimit kalim apo dështim dhe do ta rrisë numëruesin përkatës. Pra detajimi i urdhrit të mëparshëm në pseudokod është:

Përderisa numëruesi i studentëve është më i vogël ose baraz me 10

Kërko nga përdoruesi ta shkruajë rezultatin e provimit tjetër

Merre si hyrje rezultatin e provimit tjetër

Nëse studenti ka kaluar

Shtoja një kalimeve

Përndryshe

Shtoja një dështimeve

Shtoja një numëruesit të studentëve

Përdorim rreshta të zbrazët për ta izoluar urdhrin e kontrollit **if...else**, gjë që e përmirëson lexueshmërinë.

Urdhri në pseudokod:

Shfaqe përmbledhjen e rezultateve të provimit dhe vendos a duhet të marrë bonus instruktori

mund të detajohet si vijon:

Shfaqe numrin e kalimeve

Shfaqe numrin e dështimeve

Nëse kanë kaluar mbi tetë studentë

Shfaqe "Bonus për instrukturin!"

Detajimi i dytë i plotë i pseudokodit

Detajimi i dytë i plotë i pseudokodit shfaqet në figurën vijuese (5.12). Vëreni se janë përdorur rreshta të zbrazët edhe për ta dalluar urdhrin **while** për lexueshmëri. Ky pseudokod tani është detajuar mjaftueshëm për t'u shndërruar në C#. Programi që e implementon algoritmin në pseudokod dhe rezultatet mostrës janë treguar në figurën 5.13.

Fig. 5.12 | Pseudokodi për problemin e rezultateve të ekzaminimit:

1	<i>Inicializoji kalimet në zero</i>
2	<i>Inicializoji dështimet në zero</i>
3	<i>Inicializojë numëruesin e studentëve në një</i>
4	
5	<i>Përderisa numëruesi i studentëve është më i vogël ose baraz me 10</i>
6	<i> Kërko nga përdoruesi ta shkruajë rezultatin e provimit tjetër</i>
7	<i> Merre si hyrje rezultatin e provimit tjetër</i>
8	
9	<i> Nëse studenti kaloi</i>
10	<i> Shtoja një kalimeve</i>
11	<i> Përndryshe</i>
12	<i> Shtoja një dështimeve</i>
13	
14	<i> Shtoja një numëruesit të studentëve</i>
15	
16	<i>Shfaqe numrin e kalimeve</i>
17	<i>Shfaqe numrin e dështimeve</i>
18	
19	<i>Nëse kanë kaluar mbi tetë studentë</i>
20	<i>Shfaqe "Bonus për instruktorin!"</i>

Aplikacioni që e implementon algoritmin në pseudokod

Programi që e implementon algoritmin në pseudokod është treguar në Fig. 5.13.

Rreshtat

```
int numriIKalimeve = 0;  
int numriIDeshtimeve = 0;  
int numeruesiIStudenteve = 1;
```

dhe

```
int rezultati = int.Parse(Console.ReadLine());
```

i deklarojnë ndryshoret lokale që i përdor metoda Main për t'i procesuar rezultatet e ekzaminimit.


```
// Fig. 5.13: Analiza.cs
// Analiza e rezultateve të ekzaminimit
// duke i përdorur urdhrat e ndërfutur të kontrollit

using System;

class Analiza
{
    static void Main()
    {
        // inicializoj variablat (ndryshoret) në deklarime
        int numriIKalimeve = 0;
        int numriIDeshtimeve = 0;
        int numeruesiIStudenteve = 1;

        // procesoji 10 studentë duke përdorur
        // përsëritje të kontrolluar me numërues
        // (counter-controlled iteration)
        while (numeruesiIStudenteve <= 10)
        {
            // kërkoja shënimin hyrës përdoruesit dhe merre një vlerë
            Console.WriteLine("Shkruaje rezultatin (1 = kalon, 2 = dështon): ");
            int rezultati = int.Parse(Console.ReadLine());

            // if...else është ndërfutur në urdhrin while
            if (rezultati == 1)
            {
                numriIKalimeve = numriIKalimeve + 1; // rrite numrin e kalimeve
            }
            else
            {
                numriIDeshtimeve = numriIDeshtimeve + 1; // rrite nr. e dështimeve
            }

            // rrite numeruesin e studentëve ashtu që cikli dikur përfundon
            numeruesiIStudenteve = numeruesiIStudenteve + 1;
        }

        // faza e përfundimit; përgatiti dhe shfaq rezultatat
        Console.WriteLine(
            $"Kaluan: {numriIKalimeve}\nDështuan: {numriIDeshtimeve}");

        // përcaktoje se a kanë kaluar mbi 8 studentë
        if (numriIKalimeve > 8)
        {
            Console.WriteLine("Bonus për instruktorin!");
        }
    }
}
```

```

Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 2
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Kaluan: 9
Dështuan: 1
Bonus për instruktorin!

```

```

Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 2
Shkruaje rezultatin (1 = kalon, 2 = dështon): 2
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 2
Shkruaje rezultatin (1 = kalon, 2 = dështon): 2
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 1
Shkruaje rezultatin (1 = kalon, 2 = dështon): 2
Kaluan: 5
Dështuan: 5

```

Fig. 5.13 | Analiza e rezultateve të ekzaminimit, duke i përdorur urdhrat e ndërfutur të kontrollit.

Urdhri `while` e përsërit ciklin 10 herë. Gjatë secilit iterim, cikli e merr si hyrje dhe e proceson një rezultat të provimit. Vëreni se urdhri **`if...else`** për procesim të secilit rezultat është i **ndërfutur** në urdhrin **`while`**. Nëse rezultati është 1, urdhri `if...else` e rrit numrin e kalimeve; përndryshe supozon se rezultati është 2 dhe e rrit numrin e dështimeve. Rreshti i fundit brenda ciklit `while` e rrit numëruesin e studentëve para se të testohet përsëri kushti i ciklit në rreshtin

```
while (numeruesiIStudenteve <= 10)
```

Pasi të jenë regjistruar 10 vlera, cikli përfundon dhe rreshti pas trupit të `while` e shfaq numrin e kalimeve dhe numrin e dështimeve. Rreshtat vijues e përcaktojnë se a e kanë kaluar provimin mbi tetë studentë dhe, nëse po, e nxjerrin në dalje mesazhin “Bonus për instruktorin!”.

Figura 5.13 e tregon hyrjen dhe daljen nga dy ekzekutime të aplikacionit. Gjatë të parit, kushti është true (i vërtetë, i saktë) – mbi tetë studentë e kaluan provimin, prandaj aplikacioni e nxjerr një mesazh për t’i dhënë bonus instruktorit.

5.12 Operatorët e përbërë të caktimit të vlerës (Compound assignment operators)

- C# i ofron disa operatorë të përbërë të caktimit të vlerës për shkurtimin e shprehjeve të caktimit të vlerës, duke përfshirë +=, -=, *=, /= dhe %=.

Operatori i caktimit të vlerës	Shprehja shembull	Shpjegimi	la cakton
<i>Supozo: int c = 3, d = 5, e = 4, f = 6, g = 12;</i>			
+=	c += 7	c = c + 7	10 c-së
-=	d -= 4	d = d - 4	1 d-së
*=	e *= 5	e = e * 5	20 e-së
/=	f /= 3	f = f / 3	2 f-së
%=	g %= 9	g = g % 9	3 g-së

Fig. 5.14 | Operatorët e përbërë aritmetikë të caktimit të vlerës.

5.13 Operatorët e rritjes dhe zbritjes së vlerës (increment and decrement operators)

- C# e ofron operatorin unar të rritjes së vlerës, ++, dhe operatorin unar të zvogëlimit të vlerës, --, për t'ia shtuar 1 apo për t'ia hequr 1 vlerës së një variable numerike.
- Rritja (apo zvogëlimi) i një variable me inkrementin **prefiks** (rritësin parashtesë) apo dekrementin **prefiks** (zvogëluesin parashtesë) e rrit (apo e zvogëlon) variablën për 1 **dhe e përdor vlerën e re** të variablës në shprehjen në të cilën shfaqet variabla. Rritja (apo zvogëlimi) i një variable me inkrementin **postfiks** (rritësin prapashtesë) apo dekrementin **postfiks** (zvogëluesin prapashtesë) e rrit (apo e zvogëlon) variablën për 1 **por e përdor vlerën fillestare** të variablës në shprehjen në të cilën shfaqet variabla.

Operatori	Shprehja shembull	Shpjegimi
++ (inkrementi prefiks, rritësi parashtesë)	++a	E rrit a-në për 1 dhe e përdor vlerën e re të a-së në shprehjen në të cilën gjendet a-ja.
++ (inkrementi postfiks, rritësi parashtesë)	a++	E rrit a-në për 1, por e përdor vlerën fillestare (origjinale) të a-së në shprehjen në të cilën gjendet a-ja.
-- (dekrementi prefiks, zvogëluesi parashtesë)	--b	E zvogëlon b-në për 1 dhe e përdor vlerën e re të b-së në shprehjen në të cilën gjendet b-ja.
-- (dekrementi prefiks, zvogëluesi parashtesë)	b--	E zvogëlon b-në për 1 por e përdor vlerën fillestare (origjinale) të b-së në shprehjen në të cilën gjendet b-ja.

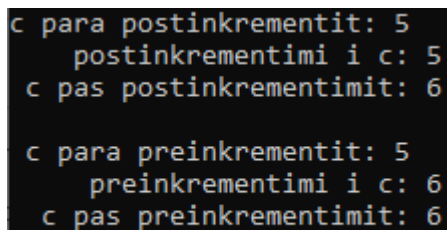
Fig. 5.15 | Operatorët rritës dhe zvogëlues.

```
// Fig. 5.15: Rritja.cs
// Operatorët inkrementi prefiks (rritësi parashtesë)
// dhe inkrementi postfiks (rritësi prapashtesë).
using System;

class Rritja
{
    static void Main()
    {
        // demonstroje operatorin postfix increment
        // (rritësi prapashtesë)
        int c = 5; // caktoja 5 c-së
        Console.WriteLine($"c para postinkrementit: {c}"); // e shfaq 5
        Console.WriteLine($"    postinkrementimi i c: {c++}"); // e shfaq 5
        Console.WriteLine($"c pas postinkrementimit: {c}"); // e shfaq 6

        Console.WriteLine(); // kaloje një rresht

        // demonstroje operatorin postfix increment
        c = 5; // caktoja 5 c-së
        Console.WriteLine($"c para preinkrementit: {c}"); // e shfaq 5
        Console.WriteLine($"    preinkrementimi i c: {++c}"); // e shfaq 6
        Console.WriteLine($"c pas preinkrementimit: {c}"); // e shfaq 6
        Console.ReadKey();
    }
}
```



```
c para postinkrementit: 5
    postinkrementimi i c: 5
c pas postinkrementimit: 6

c para preinkrementit: 5
    preinkrementimi i c: 6
c pas preinkrementimit: 6
```

5.13.3 Përparësia dhe asociativiteti i (shoqërueshmëria e) operatorëve

Operatorët	Asociativiteti (shoqërueshmëria)	Tipi
. new ++(postfix) --(postfix)	nga e majta në të djathtë	përparësia më e lartë
++ -- + - (type)	nga e djathta në të majtë	prefiksi unar (parashtesa unare)
* / %	nga e majta në të djathtë	shumëzues
+ -	nga e majta në të djathtë	mbledhës
< <= > >=	nga e majta në të djathtë	relacional
== !=	nga e majta në të djathtë	barazi
?:	nga e djathta në të majtë	kushtëzues
= += -= *= /= %=	nga e djathta në të majtë	caktim i vlerës

Fig. 5.17 | Përparësia dhe asociativiteti i (shoqërueshmëria e) operatorëve të diskutuar deri tash.

5.14 Tipet e thjeshta

- C# kërkon që të gjitha variablat të kenë tip.
- Variablat e tipeve të thjeshta të deklaruara jashtë një metode si fusha (fields) të një klase – automatikisht u caktohen vlerat e paracaktuara. Variablat e instancës të tipeve **char**, **byte**, **sbyte**, **short**, **ushort**, **int**, **uint**, **long**, **ulong**, **float**, **double** dhe **decimal** të gjithave u jepet vlera 0 fillimisht (by default). Variablat e instancës të tipit **bool** u jepet vlera **false** fillimisht. Variablat e instancës të tipit referencë inicializohen fillimisht me vlerën **null**.

6. Urdhrat e kontrollit: Pjesa 2

6.2 Bazat e përsëritjes së (iterimit të) kontrolluar nga numëruesi (Counter-Controlled Iteration)

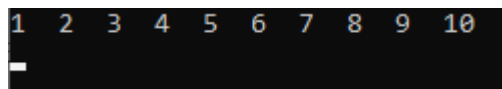
- Përsëritja e kontrolluar nga numëruesi e kërkon një variabël të kontrollit, vlerën fillestare të ndryshores së kontrollit, rritjen (inkrementin) apo zvogëlimin (dekrementin) me të cilin ndryshorja e kontrollit ndryshohet (modifikohet) secilën herë nëpër cikël (unazë, loop), dhe kushtin për vazhdim të ciklit që e përcakton se a duhet të vazhdojë përsëritja në cikël.

```
// Fig. 6.1: NumeruesMeWhile.cs
// Përsëritje e kontrolluar nga numëruesi
// me urdhrin e iterimit (përsëritjes) while
using System;

class NumeruesMeWhile
{
    static void Main()
    {
        int numeruesi = 1; // deklarohet dhe inicializohet variabëlën e kontrollit

        while (numeruesi <= 10) // kushti për vazhdimin e ciklit
        {
            Console.Write($"{numeruesi} ");
            ++numeruesi; // rrite variabëlën e kontrollit
        }

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



```
1 2 3 4 5 6 7 8 9 10
_
```

6.3 Urdhri i përsëritjes (iterimit) *for*

- Zakonisht, urdhrat **for** përdoren për përsëritje të kontrolluar me numërues (iterim definitiv), dhe urdhrat **while** për përsëritje të kontrolluar nga vlera vëzhguese (iterim indefinitiv).
- Ballina e (header-i i) urdhrit **for** “i bën të gjitha”: e specifikon secilin nga elementet e nevojshme për përsëritjen e kontrolluar me numërues me një ndryshore të kontrollit.
- Format i përgjithshëm për urdhrin **for** është

```
for (inicializimi; kushtiPërVazhdiminECiklit; rritja)  
{  
    urdhri;  
}
```

ku shprehja *inicializimi* e emëron ndryshoren e kontrollit të ciklit dhe e jep vlerën e saj fillestare, *kushtiPërVazhdiminECiklit* është kushti që e përcakton se a duhet të vazhdojë përsëritja në cikël (looping) dhe *rritja* (inkrementi) e ndryshon vlerën e ndryshores së kontrollit ashtu që kushti i vazhdimit të ciklit dikur bëhet **false**.

- Skopi (fushëveprimi, shtrirja, scope) i ndryshores e përcakton se ku mund të përdoret ajo në aplikacion. Ndryshorja lokale mund të përdoret vetëm në metodën që e deklaron ndryshoren dhe vetëm nga pika e deklarimit deri në fund të bllokut në të cilin është deklaruar ndryshorja.
- Rritja (inkrementi) për një urdhër **for** mund të jetë edhe negativ-e, me ç'rast është zbritje (dekrement) dhe cikli numëron teposhtë.


```
// Fig. 6.2: NumeruesMeFor.cs
// Përsëritje e kontrolluar nga numëruesi
// me urdhrin e iterimit (përsëritjes) for.
using System;

class NumeruesMeFor
{
    static void Main()
    {
        // ballina e (header-i i) urdhrit for
        // e përfshin inicializimin,
        // kushtin për vazhdimin e ciklit dhe rritjen
        for (int numeruesi = 1; numeruesi <= 10; ++numeruesi)
        {
            Console.Write($"{numeruesi} ");
        }

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

```
1 2 3 4 5 6 7 8 9 10
_
```

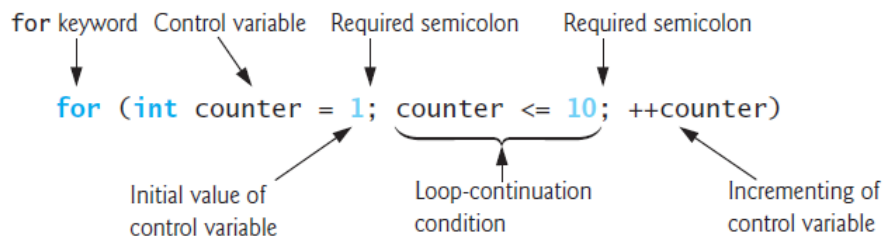


Fig. 6.3 | `for` statement header components.

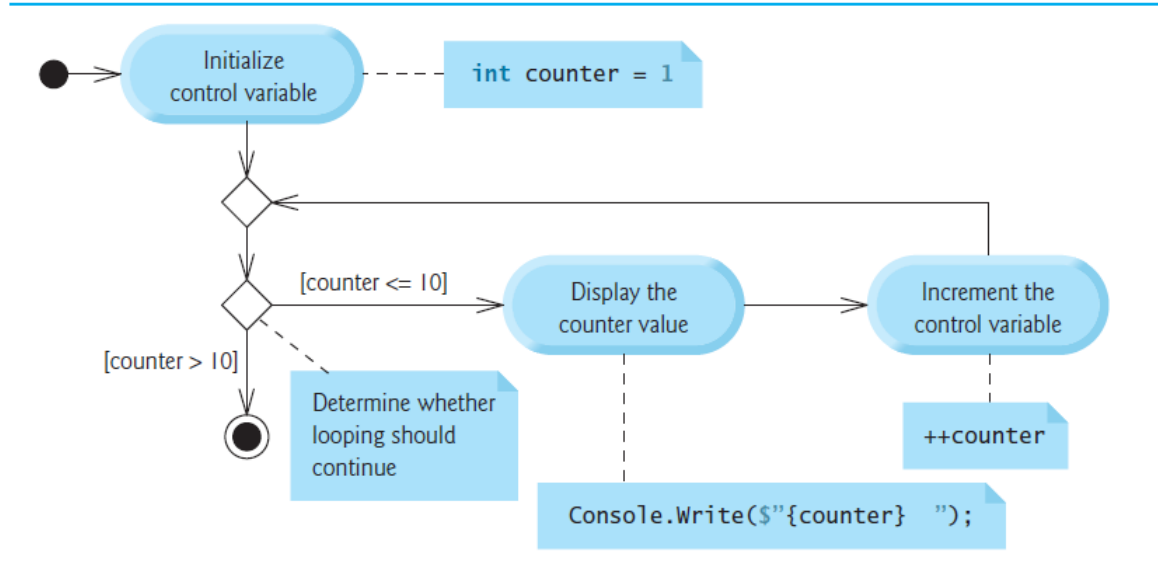


Fig. 6.4 | UML activity diagram for the for statement in Fig. 6.2.

6.5 Aplikacion: Mbledhja e numrave çiftë

```

// Fig. 6.5: Shuma.cs
// Mbledhja e numrave të plotë me urdhrin for.
using System;

class Shuma
{
    static void Main()
    {
        int totali = 0; // inicializoje totalin

        // mbledhi (totalizoji) numrat çiftë nga 2 deri në 20
        for (int numri = 2; numri <= 20; numri += 2)
        {
            totali += numri;
        }

        Console.WriteLine($"Shuma është {totali}"); // shfaqje rezultatin
        Console.ReadKey();
    }
}
  
```

```

Shuma është 110
_
  
```

6.6 Aplikacion: Llogaritjet e interesit të përbërë

- Kur një ndryshore e tipit **decimal** inicializohet me vlerë **int**, vlera e tipit **int** promovohet në (“ngritet në pozitën”) tip **decimal** në mënyrë implicite – nuk kërkohet shndërrim (cast).
- Metodat që duhet të thirren duke e përdorur emrin e klasës quhen metoda statike (**static**).
- C# nuk përfshin operator të fuqisë matematike. Në vend të kësaj, **Math.Pow(x, y)** e llogarit vlerën e x të ngritur në fuqinë e y -të. Metoda i pranon dy argumente **double** dhe kthen vlerë **double**.
- C# nuk do ta konvertojë (shndërrojë) në mënyrë implicite një **double** në tip **decimal**, apo anasjelltas, për shkak të humbjes së mundshme të informacionit në cilindo shndërrim. Për ta bërë këtë shndërrim, kërkohet një operator i shndërrimit (cast).
- Vlerat në një fushë (field) janë të radhitura djathtas (right-aligned) në mënyrë të paracaktuar (by default). Për të treguar se vlerat duhet të nxirren të radhitura majtas, thjesht përdor gjerësi negative të fushës.
- Në një element të formatit, numri n pas presjes tregon se vlera që nxirret duhet të shfaqet me gjerësi të fushës n – domethënë, me të paktën n pozicione të karaktereve, duke i përdorur hapësirat si karaktere mbushëse (fill characters).
- Numrat me pikë lëvizëse të tipit **double** (apo **float**) mund të shkaktojnë telashe në llogaritje monetare; në vend të tyre përdore tipin **decimal**.

```
// Fig. 6.6: Interesi.cs
// Llogaritjet e interesit të përbërë me urdhrin for.
using System;

class Interesi
{
    static void Main()
    {
        decimal kapitali = 1000; // sasia fillestare para interesit
        double norma = 0.05; // norma e interesit

        // shfaq titujt
        Console.WriteLine("Viti   Sasia në depozitë");

        // llogarite sasinë në depozitë për secilin nga dhjetë vitet
        for (int viti = 1; viti <= 10; ++viti)
        {
            // llogarite sasinë e re për vitin e specifikuar
            decimal sasia = kapitali *
                ((decimal)Math.Pow(1.0 + norma, viti));

            // shfaq vitin dhe sasinë
            Console.WriteLine($"{viti,4}{sasia,20:C}");
        }
        Console.ReadKey();
    }
}
```

```
Viti   Sasia në depozitë
1       £1,050.00
2       £1,102.50
3       £1,157.63
4       £1,215.51
5       £1,276.28
6       £1,340.10
7       £1,407.10
8       £1,477.46
9       £1,551.33
10      £1,628.89
```

6.7 Urdhri i përsëritjes (iterimit) *do...while*

- Urdhri **do...while** e teston kushtin për vazhdim të ciklit *pas* ekzekutimit të trupit të ciklit; prandaj, gjithmonë trupi ekzekutohet të paktën një herë.

- Urdhri **do...while** e ka formën

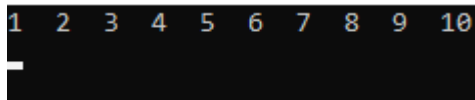
```
do
{
    urdhri;
} while (kushti);
```

```
// Fig. 6.7: TestIDoWhile.cs
// urdhri i iterimit (përsëritjes) do...while.
using System;

class TestIDoWhile
{
    static void Main()
    {
        int numeruesi = 1; // inicializojë numeruesin

        do
        {
            Console.Write($"{numeruesi} ");
            ++numeruesi;
        } while (numeruesi <= 10); // pikëpresja duhet

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



```
1 2 3 4 5 6 7 8 9 10
_
```

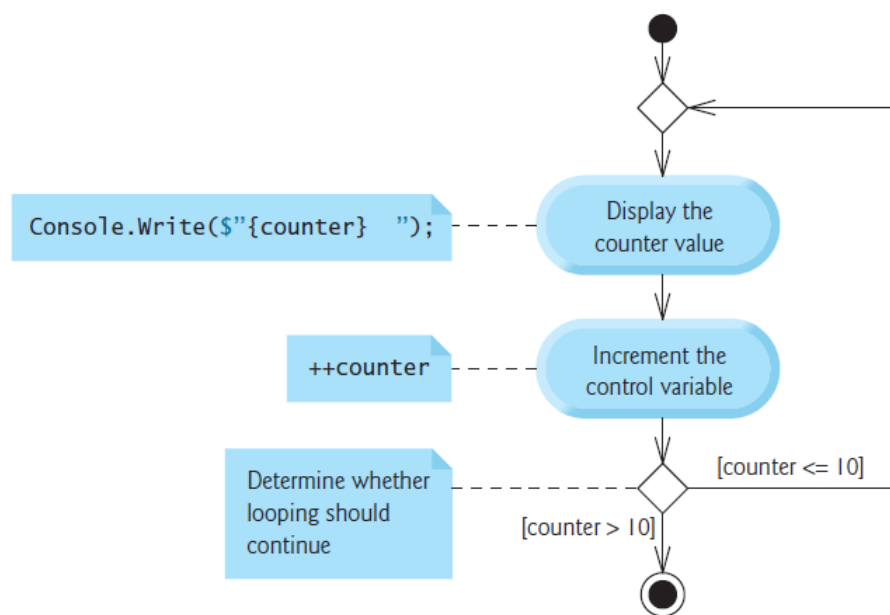


Fig. 6.8 | `do...while` iteration statement UML activity diagram.

6.8 Urdhri *switch* me përzgjedhje të shumëfishtë

- Urdhri **switch** me përzgjedhje të shumëfishtë kryen veprime të ndryshme bazuar në vlerat e mundshme të një shprehjeje.
- Metoda **Console.ReadLine** kthen **null** kur haset sekuenca e tasteve "<Ctrl> Z" që interpretohet si end-of-file (fundi-i-fajllit).
- Blloku i urdhrit **switch** e përmban një sekuencë të etiketave (labelave) **case** dhe një etiketë opsionale **default**.
- Shprehja në kllapa që e pason fjalën kyçe **switch** është shprehja **switch**. Aplikacioni tenton ta përputhë vlerën e shprehjes **switch** me një etiketë **case**. Nëse ndodh përputhja, aplikacioni i ekzekuton urdhrat për atë rast (**case**).
- Urdhri **case** nuk ofron mekanizëm për të testuar rangje të vlerave, kështu që çdo vlerë që duhet të testohet duhet të renditet në një etiketë të veçantë **case**.
- Pasi të ekzekutohen urdhrat në një case (rast), kërkohet ta përfshini një urdhër që e përfundon atë rast, si një **break** apo një **return**.
- Nëse nuk ndodh asnjë përputhje midis vlerës së shprehjes dhe një etikete **case**, ekzekutohen urdhrat pas etiketës **default**. Nëse nuk ndodh asnjë përputhje dhe **switch**-i nuk përmban etiketë **default**, kontrolli i programit vazhdon me urdhrin e parë pas urdhrit switch.
- Urdhri **break** shkakton dalje të menjëhershme nga një urdhër **while**, **for**, **do...while**, **switch** apo **foreach**. Ekzekutimi vazhdon me urdhrin e parë pas urdhrit të kontrollit.
- Urdhri **continue**, kur ekzekutohet në një **while**, **for**, **do...while** apo **foreach**, i anashkalon urdhrat e mbetur në trupin e ciklit dhe vazhdon me iterimin (përsëritjen) tjetër në cikël. Në një urdhër **for**, rritja (inkrementi) kryhet para se të testohet kushti për vazhdimin e ciklit.

```

// Fig. 6.9: DitariINotave.cs
// Përdorja e një urdhri switch për t'i numëruar
// notat shkronjore.
using System;

class DitariINotave
{
    static void Main()
    {
        int totali = 0; // shuma e notave
        int numeruesiINotave = 0; // numri i notave të shkruara
        int numriIAve = 0; // numri i notave A
        int numriIBve = 0; // numri i notave B
        int numriICve = 0; // numri i notave C
        int numriIDve = 0; // numri i notave D
        int numriIFve = 0; // numri i notave F

        Console.WriteLine("Shkruaji notat si numra të plotë në rangun (intervalin) 0-100.");
        Console.WriteLine(
            "Shtyp <Ctrl> z dhe pastaj Enter për ta përfunduar futjen e notave:");

        string hyrja = Console.ReadLine(); // lexoje hyrjen nga përdoruesi

        // sillu në cikël deri kur përdoruesi e fut treguesin (indiktorin)
        // end-of-file (fundi i fajllit) (<Ctrl> z)
        while (hyrja != null)
        {
            int nota = int.Parse(hyrja); // lexoje notën nga hyrja e përdoruesit
            totali += nota; // shtoja notën totalit
            ++numeruesiINotave; // rrite numrin e notave

            // përcaktoje se cila notë është futur
            switch (nota / 10)
            {
                case 9: // nota ishte në 90-tat
                case 10: // nota ishte 100
                    ++numriIAve; // rrite numriIAve
                    break; // e nevojshme për të dalë nga switch-i
                case 8: // nota ishte ndërmjet 80 e 89
                    ++numriIBve; // rrite numriIBve
                    break; // dil nga switch-i
                case 7: // nota ishte ndërmjet 70 e 79
                    ++numriICve; // rrite numriICve
                    break; // dil nga switch-i
                case 6: // nota ishte ndërmjet 60 e 69
                    ++numriIDve; // rrite numriIDve
                    break; // dil nga switch-i
                default: // nota ishte më pak se 60
                    ++numriIFve; // rrite numriIFve
                    break; // dil nga switch-i
            }

            hyrja = Console.ReadLine(); // lexoje hyrjen nga përdoruesi
        }
    }
}

```



```

Console.WriteLine("\nRaporti i notave:");

// nëse përdoruesi e shkroi të paktën një notë...
if (numeruesiINotave != 0)
{
    // llogarite mesataren e të gjitha notave të futura
    double mesatarja = (double)totali / numeruesiINotave;

    // afishoje (nxirre në dalje) përmbledhjen e rezultateve
    Console.WriteLine(
        $"Totali i {numeruesiINotave} notave të futura është {totali}");
    Console.WriteLine($"Mesatarja e klasës është {mesatarja:F}");
    Console.WriteLine("Numri i studentëve sipas notave:");
    Console.WriteLine($"A: {numriIAve}"); // shfaq numrin e notave A
    Console.WriteLine($"B: {numriIBve}"); // shfaq numrin e notave B
    Console.WriteLine($"C: {numriICve}"); // shfaq numrin e notave C
    Console.WriteLine($"D: {numriIDve}"); // shfaq numrin e notave D
    Console.WriteLine($"F: {numriIFve}"); // shfaq numrin e notave F
}
else // asnjë notë nuk është futur, prandaj shfaq mesazhin përkatës
{
    Console.WriteLine("Asnjë notë nuk është futur");
}
Console.ReadKey();
}
}

```

```

Shkruaji notat si numra të plotë në rangun (intervalin) 0-100.
Shtyp <Ctrl> z dhe pastaj Enter për ta përfunduar futjen e notave:
99
92
45
57
63
71
76
85
90
100
^Z

Raporti i notave:
Totali i 10 notave të futura është 778
Mesatarja e klasës është 77.80
Numri i studentëve sipas notave:
A: 4
B: 1
C: 2
D: 1
F: 2

```

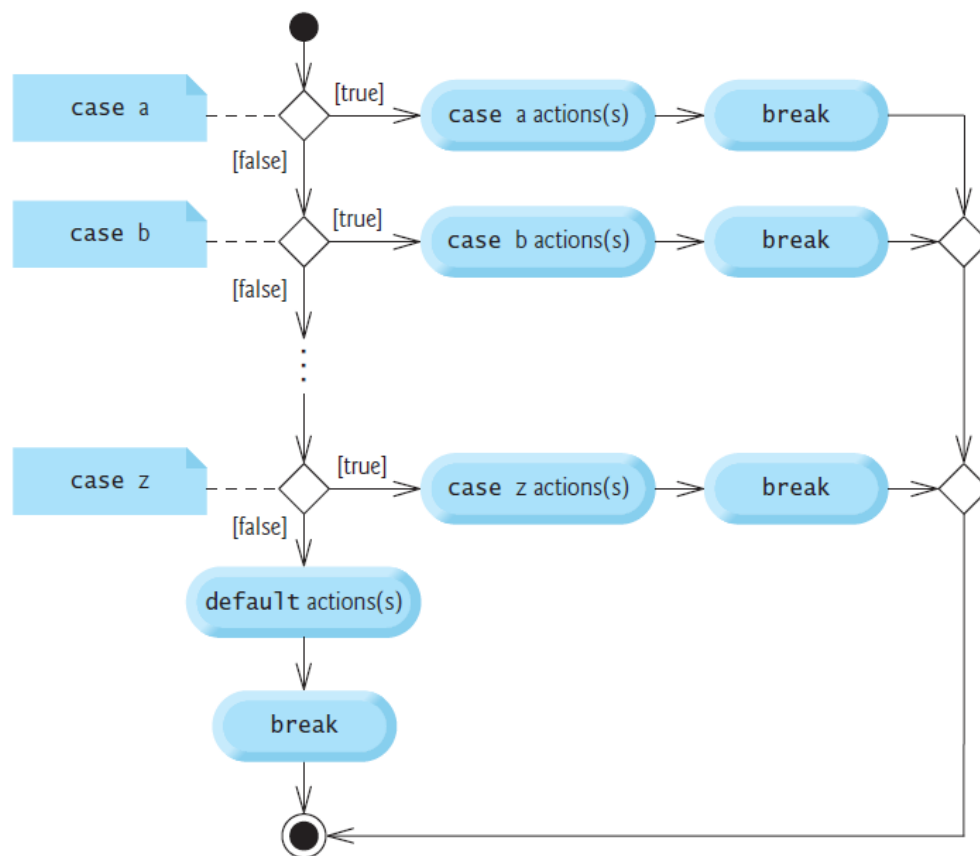


Fig. 6.10 | switch multiple-selection statement UML activity diagram with break statements.

6.9 Studimi i rastit Klasa PoliseEVetures: stringjet në urdhra switch

```
// Fig. 6.11: PoliseEVetures.cs
// Klasë që e paraqet një polisë të sigurimit të veturës.
class PoliseEVetures
{
    public int NumriILlogarise { get; set; } // numri i llogarisë së polisës
    public string MarkaDheModeli { get; set; } // vetura së cilës i aplikohet polisa
    public string Rajoni { get; set; } // shkurtesa dy-shkronjëshe për rajon (apo shtet)

    // konstruktori
    public PoliseEVetures(int numriILlogarise, string markaDheModeli, string rajoni)
    {
        NumriILlogarise = numriILlogarise;
        MarkaDheModeli = markaDheModeli;
        Rajoni = rajoni;
    }

    // përgjigjet (kthen) se a ka rajoni sigurim pa-faj (no-fault)
    public bool EshteRajonPaFaj
    {
        get
        {
            bool rajonPaFaj;

            // përcaktoje se a ka rajoni sigurim të veturës pa faj
            switch (Rajoni) // merre shkurtesën e Rajonit të objektit PoliseEVetures
            {
                case "MA":
                case "NJ":
                case "NY":
                case "PA":
                    rajonPaFaj = true;
                    break;
                default:
                    rajonPaFaj = false;
                    break;
            }

            return rajonPaFaj;
        }
    }
}
```

```
// Fig. 6.12: TestIPolisesSeVetures.java
// Demonstrimi i stringjeve në switch.
using System;

class TestIPolisesSeVetures
{
    static void Main()
    {
        // krijoji dy objekte PoliseEVetures
        PoliseEVetures polisa1 = new PoliseEVetures(11111111, "Toyota Camry", "NJ");
        PoliseEVetures polisa2 = new PoliseEVetures(22222222, "Ford Fusion", "ME");

        // shfaq a është secila polisë në rajon pa-faj
        PolisaNeRajonPaFaj(polisa1);
        PolisaNeRajonPaFaj(polisa2);
        Console.ReadKey();
    }

    // metodë që e shfaq se a është një PoliseEVetures
    // në rajon me sigurim të polisës pa-faj
    public static void PolisaNeRajonPaFaj(PoliseEVetures polisa)
    {
        Console.WriteLine("Polisa e veturës:");
        Console.WriteLine($"Llogaria #: {polisa.NumriILlogarise}; ");
        Console.WriteLine($"Vetura: {polisa.MarkaDheModeli}; ");
        Console.WriteLine($"Rajoni {polisa.Rajoni} ");
        Console.WriteLine($"{{(polisa.EshteRajonPaFaj ? " është" : " nuk është")}}");
        Console.WriteLine(" rajon pa-faj (no-fault)\n");
    }
}
```

```
Polisa e veturës:
Llogaria #: 11111111; Vetura: Toyota Camry;
Rajoni NJ është rajon pa-faj (no-fault)

Polisa e veturës:
Llogaria #: 22222222; Vetura: Ford Fusion;
Rajoni ME nuk është rajon pa-faj (no-fault)
```

6.10 Urdhrrat break dhe continue

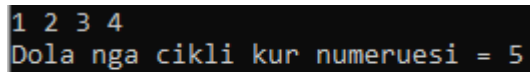
```
// Fig. 6.13: TestIBreak.cs
// urdhri break për dalje nga një urdhër for.
using System;

class TestIBreak
{
    static void Main()
    {
        int numeruesi; // variabël e kontrollit që përdoret edhe pasi të mbaron cikli

        for (numeruesi = 1; numeruesi <= 10; ++numeruesi) // sillu në cikël 10 herë
        {
            if (numeruesi == 5) // nëse numeruesi është 5,
            {
                break; // përfundoje ciklin
            }

            Console.Write($"{numeruesi} ");
        }

        Console.WriteLine($"\\nDola nga cikli kur numeruesi = {numeruesi}");
        Console.ReadKey();
    }
}
```



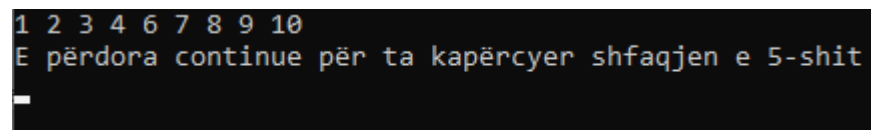
```
1 2 3 4
Dola nga cikli kur numeruesi = 5
_
```

```
// Fig. 6.14: TestIContinue.cs
// urdhri continue që e kapërcen një interacion për një urdhër for.
using System;

class TestIContinue
{
    static void Main()
    {
        for (int numeruesi = 1; numeruesi <= 10; ++numeruesi) // loop 10 times
        {
            if (numeruesi == 5) // nëse numeruesi është 5,
            {
                continue; // kapërceje (kaloje) kodin e mbetur në cikël
            }

            Console.Write($"{numeruesi} ");
        }

        Console.WriteLine("\nE përdora continue për ta kapërcyer shfaqjen e 5-shit");
        Console.ReadKey();
    }
}
```



```
1 2 3 4 6 7 8 9 10
E përdora continue për ta kapërcyer shfaqjen e 5-shit
_
```

6.11 Operatorët logjikë

- Operatorët logjikë ju mundësojnë të formoni kushte më komplekse (më të përbëra) duke i kombinuar kushte të thjeshta. Operatorët logjikë janë **&&** (DHE, AND i kushtëzuar), **||** (OSE, OR i kushtëzuar), **&** (DHE, AND logjik bulean), **|** (OSE, OR përfshirës/inkluziv logjik bulean), **^** (OSE, OR përjashtues/ekskluziv logjik bulean) dhe **!** (negacioni, mohimi logjik).
- Operatori **&&** (DHE, AND i kushtëzuar) siguron që kushtet janë të *dy* **true** (të vërteta) para se ta zgjedhë një rrugë të caktuar të ekzekutimit.
- Operatori **||** (OSE, OR i kushtëzuar) siguron që *njëri apo të dy* kushtet janë **true** (të vërteta) para se ta zgjedhë një rrugë të caktuar të ekzekutimit.
- Pjesët e një shprehjeje që përmbajnë operatorë **&&** apo **||** vlerësohen vetëm deri kur të dihet se a është kushti **true** (i vërtetë, i saktë) apo **false** (i pavërtetë, i pasaktë). Kjo veçori e AND-it kushtëzuar dhe e OR-it kushtëzuar quhet vlerësim i shkurtuar (**short-circuit evaluation**, vlerësim me “qark të shkurtër”).
- Operatori logjik bulean AND (**&**, DHE) dhe operatori OR (OSE, **|**) përfshirës (inkluziv) logjik bulean punojnë në mënyrë identike me operatorët **&&** (AND, DHE kushtëzues) dhe **||** (OR, OSE kushtëzues), por operatorët logjikë buleanë gjithmonë i vlerësojnë të dy operandët e tyre (domethënë nuk bëjnë vlerësim të shkurtuar - short-circuit evaluation, vlerësim me “qark të shkurtër”).
- Një shprehje komplekse (e përbërë) që e përmban operatorin përjashtues (ekskluziv) logjik bulean **OR** (OSE, **^**) është **true** (e vërtetë, e saktë) nëse dhe vetëm nëse (atëherë dhe vetëm atëherë kur) njëri nga operandët e saj është **true** dhe tjetri është **false**. Nëse të dy operandët janë **true** apo të dytë janë **false**, i tërë kushti është **false**.
- Operatori **!** (negacioni, mohimi logjik) jua mundëson ta “përmbysni” (“ta minusoni”, “ta ktheni mbrapsht”) kuptimin e një kushti. Operatori logjik i negacionit vendoset para një kushti për ta zgjedhur një rrugë të ekzekutimit nëse kushti fillestar është **false**. Në shumicën e rasteve, mund ta shmangni përdorimin e negacionit logjik duke e shprehur kushtin ndryshe me një operator të duhur relacional apo të barazisë.

expression1	expression2	expression1 && expression2
false	false	false
false	true	false
true	false	false
true	true	true

Fig. 6.15 | && (conditional AND) operator truth table.

expression1	expression2	expression1 expression2
false	false	false
false	true	true
true	false	true
true	true	true

Fig. 6.16 | || (conditional OR) operator truth table.

expression1	expression2	expression1 ^ expression2
false	false	false
false	true	true
true	false	true
true	true	false

Fig. 6.17 | ^ (boolean logical exclusive OR) operator truth table.

expression	!expression
false	true
true	false

Fig. 6.18 | ! (logical negation) operator truth table.


```

// Fig. 6.19: OperatoretLogjike.cs
// Operatorët logjikë.
using System;

class OperatoretLogjike
{
    static void Main()
    {
        // krijoje tabelën e vërtetësisë për operatorin && (DHE i kushtëzuar)
        Console.WriteLine("DHE i kushtëzuar (&&)");
        Console.WriteLine($"false && false: {false && false}");
        Console.WriteLine($"false && true: {false && true}");
        Console.WriteLine($"true && false: {true && false}");
        Console.WriteLine($"true && true: {true && true}\n");

        // krijoje tabelën e vërtetësisë për operatorin || (OSE i kushtëzuar)
        Console.WriteLine("OSE i kushtëzuar (||)");
        Console.WriteLine($"false || false: {false || false}");
        Console.WriteLine($"false || true: {false || true}");
        Console.WriteLine($"true || false: {true || false}");
        Console.WriteLine($"true || true: {true || true}\n");

        // krijoje tabelën e vërtetësisë për operatorin & (DHE logjik bulean)
        Console.WriteLine("DHE logjik bulean (&)");
        Console.WriteLine($"false & false: {false & false}");
        Console.WriteLine($"false & true: {false & true}");
        Console.WriteLine($"true & false: {true & false}");
        Console.WriteLine($"true & true: {true & true}\n");

        // krijoje tabelën e vërtetësisë për operatorin |
        // (OSE përfshirës (inkluziv) logjik bulean)
        Console.WriteLine("OSE përfshirës (inkluziv) logjik bulean (|)");
        Console.WriteLine($"false | false: {false | false}");
        Console.WriteLine($"false | true: {false | true}");
        Console.WriteLine($"true | false: {true | false}");
        Console.WriteLine($"true | true: {true | true}\n");

        // krijoje tabelën e vërtetësisë për operatorin ^
        // (OSE përjashtues (ekskluziv) logjik bulean)
        Console.WriteLine("OSE përjashtues (ekskluziv) logjik bulean (^)");
        Console.WriteLine($"false ^ false: {false ^ false}");
        Console.WriteLine($"false ^ true: {false ^ true}");
        Console.WriteLine($"true ^ false: {true ^ false}");
        Console.WriteLine($"true ^ true: {true ^ true}\n");

        // krijoje tabelën e vërtetësisë për operatorin !
        // (negacioni (mohimi) logjik)
        Console.WriteLine("Negacioni (mohimi) logjik (!)");
        Console.WriteLine($"!false: {!false}");
        Console.WriteLine($"!true: {!true}");
        Console.ReadKey();
    }
}

```

```
DHE i kushtëzuar (&&)
false && false: False
false && true: False
true && false: False
true && true: True

OSE i kushtëzuar (||)
false || false: False
false || true: True
true || false: True
true || true: True

DHE logjik bulean (&)
false & false: False
false & true: False
true & false: False
true & true: True

OSE përfshirës (inkluziv) logjik bulean (|)
false | false: False
false | true: True
true | false: True
true | true: True

OSE përjashtues (ekskluziv) logjik bulean
false ^ false: False
false ^ true: True
true ^ false: True
true ^ true: False

Negacioni (mohimi) logjik (!)
!false: True
!true: False
```

Operatorët	Asociativiteti (shoqërueshmëria)	Tipi
. new ++(postfix) --(postfix)	nga e majta në të djathtë	përparësia më e lartë
++ -- + - ! (type)	nga e djathta në të majtë	prefiksi unar (parashtesa unare)
* / %	nga e majta në të djathtë	shumëzues
+ -	nga e majta në të djathtë	mbledhës
< <= > >=	nga e majta në të djathtë	relacional
== !=	nga e majta në të djathtë	barazi
&	nga e majta në të djathtë	DHE logjik bulean
^	nga e majta në të djathtë	OSE përjashtues (ekskluziv) logjik bulean
	nga e majta në të djathtë	OSE përfshirës (inkluziv) logjik bulean
&&	nga e majta në të djathtë	DHE i kushtëzuar
	nga e majta në të djathtë	OSE i kushtëzuar
?:	nga e djathta në të majtë	kushtëzues
= += -= *= /= %=	nga e djathta në të majtë	caktim i vlerës

Fig. 6.20 | Përparësia dhe asociativiteti i (shoqërueshmëria e) operatorëve të diskutuar deri tash.

6.12 Përmbledhje e Programimit-Të-Strukturuar

- Çdo formë e kontrollit që nevojitet ndonjëherë në një aplikacion C# mund të shprehet në terma të **sekuencës** (radhitjes), urdhrit if (**përzgjedhjes**) dhe urdhrit while (**përsëritjes**). Këto mund të kombinohen në vetëm dy mënyra – **stivimi** (**stacking**, vendosja e elementeve njëri-mbi-tjetrin) dhe **ndërfutja** (**nesting**, futja e një elementi brenda një tjetri).

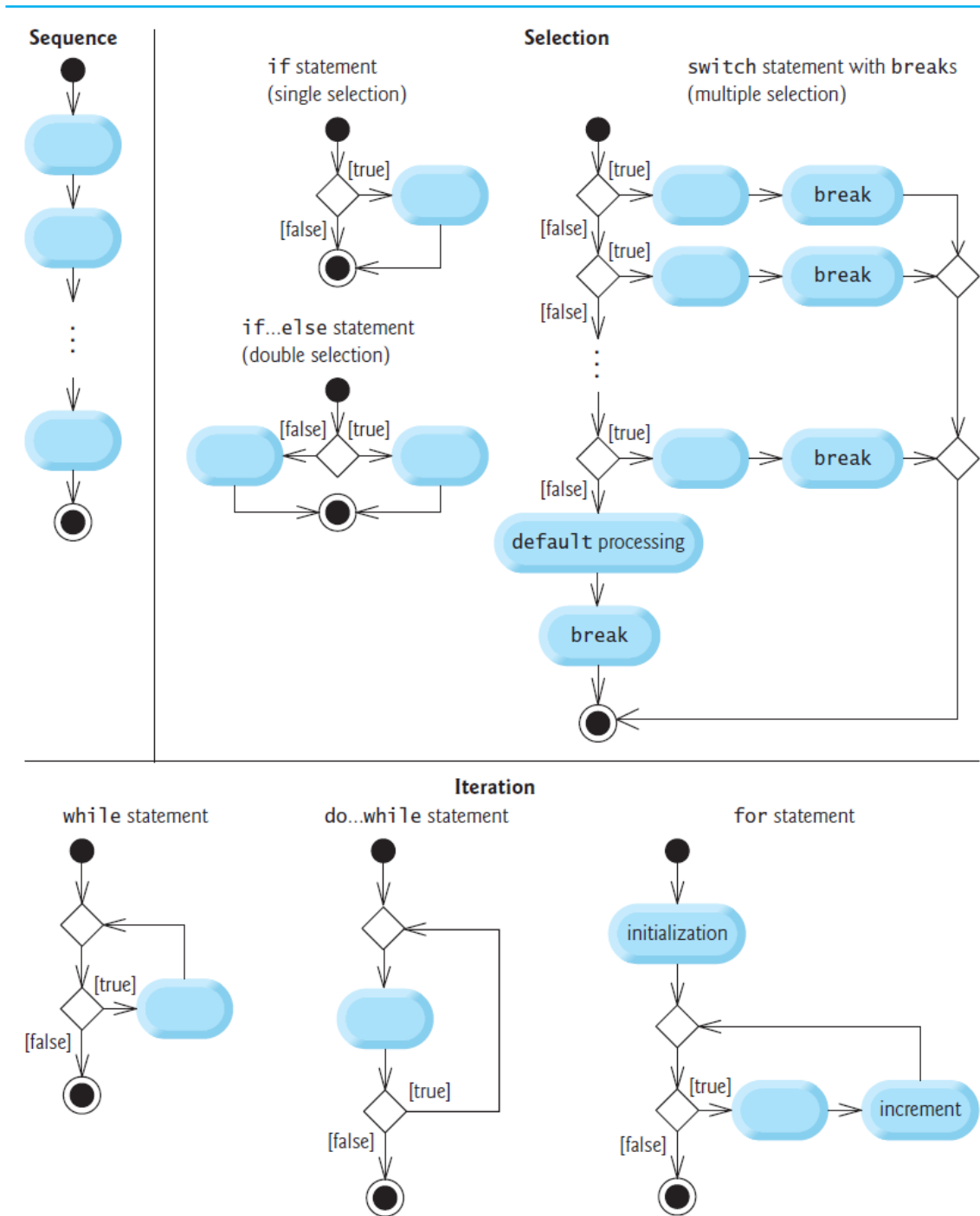
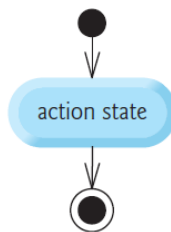
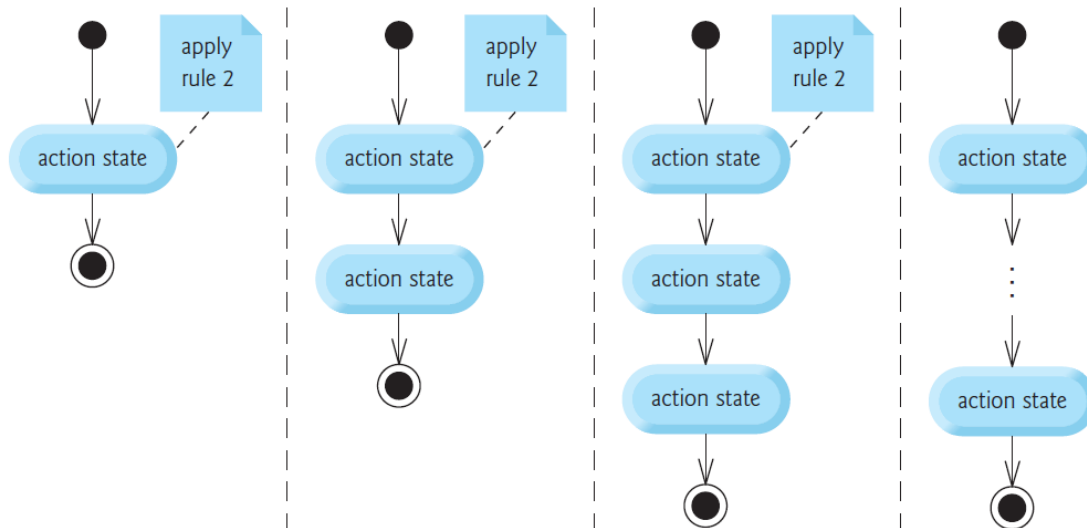


Fig. 6.21 | C#'s sequence, selection and iteration statements.

Rregullat për formimin e aplikacioneve të strukturuar

- 1 Fillo me diagramin më të thjeshtë të aktivitetit (Fig. 6.23)
- 2 Cilado gjendje e veprimit (action state) mund të zëvendësohet me dy gjendje të veprimit në sekuencë.
- 3 Cilado gjendje e veprimit mund të zëvendësohet me cilindo urdhër të kontrollit (sekuencë të gjendjeve të veprimit, if, if...else, switch, while, do...while, for apo foreach, të cilin do ta shohim në Kapitullin 8).
- 4 Rregullat 2 dhe 3 mund të zbatohen aq herë sa të jetë e nevojshme në çfarëdo renditje.

**Fig. 6.23** | Simplest activity diagram.**Fig. 6.24** | Repeatedly applying the stacking rule (Rule 2) of Fig. 6.22 to the simplest activity diagram.

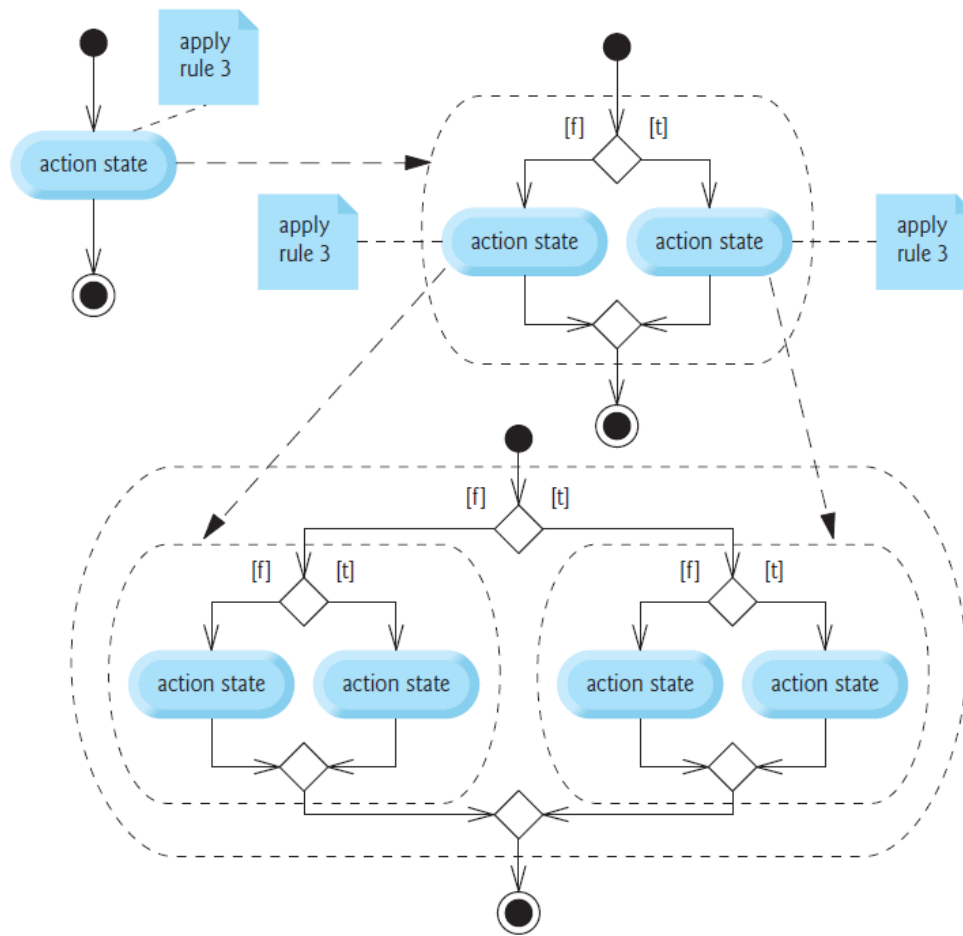


Fig. 6.25 | Repeatedly applying Rule 3 of Fig. 6.22 to the simplest activity diagram.

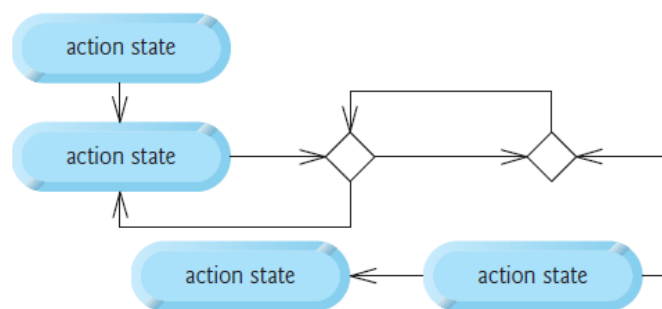


Fig. 6.26 | “Unstructured” activity diagram.

Për botimin

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/>