

# C#

## למתכנתי Java/C++/Visual C++

יש להתעלם מכל מה שקשור לתקליטור המצורף

הקבצים נמצאים באתר הוד-עמי

[www.hod-ami.co.il](http://www.hod-ami.co.il)

בקטגוריה "קבצי תרגול לספרים" יש לבחור

ב-59327 כדי להוריד את הקבצים למחשב

עורך ראשי, עריכה מקצועית: **זהר עמיהוד**

תרגום: **איציק מנשרוף**

ייעוץ מקצועי: **ארז קורן**

עריכה לשונית ועיצוב: **רמה שנקלר**

עיצוב עטיפה: **שרון רז**

## שמות מסחריים

שמות המוצרים והשירותים המוזכרים בספר הינם שמות מסחריים רשומים של החברות שלהם. הוצאת הוד-עמי ו-Microsoft Press עשו כמיטב יכולתן למסור מידע אודות השמות המסחריים המוזכרים בספר זה ולציין את שמות החברות, המוצרים והשירותים. שמות מסחריים רשומים ( registered trademarks) המוזכרים בספר צוינו בהתאמה.

**C#** הינו מוצר רשום של חברת **Microsoft**.

## הודעה

ספר זה מיועד לתת מידע אודות מוצרים שונים. נעשו מאמצים רבים לגרום לכך שהספר יהיה שלם ואמין ככל שניתן, אך אין משתמעת מכך כל אחריות שהיא.

המידע ניתן "כמות שהוא" ("as is"). הוצאת הוד-עמי ו-Microsoft Press אינן אחראיות כלפי יחיד או ארגון עבור כל אובדן או נזק אשר ייגרם, אם ייגרם, מהמידע שבספר זה, או מהתקליטור שמצורף לו.

**לשם שטף הקריאה כתוב ספר זה בלשון זכר בלבד. ספר זה מיועד לגברים ונשים כאחד ואין בכוונתנו להפלות או לפגוע בציבור המשתמשים/ות.**

☐ טלפון: 09-9564716

☐ פקס: 09-9571582

☐ דואר אלקטרוני: [info@hod-ami.co.il](mailto:info@hod-ami.co.il)

☐ אתר באינטרנט: [www.hod-ami.co.il](http://www.hod-ami.co.il)

# C#

למתכנתי

Java/C++/Visual C++

Tom Archer

**Microsoft**<sup>®</sup>



# Inside C#

By Tom Archer

Editor: **Z. Amihud**

Copyright 2001 by Microsoft Corporation.

Original English language edition Copyright © 2001 by Microsoft Corporation.

All rights published by arrangement with the original publisher, Microsoft Press, a division of Microsoft Corporation, Redmond, Washington, USA.



כל הזכויות שמורות

**הוצאת הוד-עמי**

**לספרי מחשבים בע"מ**

ת.ד. 6108 הרצליה 46160

טלפון: 09-9564716 פקס: 09-9571582

**info@hod-ami.co.il**

אין להעתיק או לשדר בכל אמצעי שהוא ספר זה או קטעים ממנו בשום צורה ובשום אמצעי אלקטרוני או מכני, לרבות צילום והקלטה, אמצעי אחסון והפצת מידע, ללא אישור בכתב מאת ההוצאה, אלא לשם ציטוט קטעים קצרים בציון שם המקור.

הודפס בישראל 2002

All Rights Reserved

**HOD-AMI Ltd.**

P.O.B. 6108, Herzliya

ISRAEL, 2002

מסת"ב 965-361-296-4 ISBN

# תוכן עניינים מקוצר

13	מבוא
17	הקדמה
<b>21</b>	<b>חלק 1 - הנחת היסודות</b>
23	פרק 1: יסודות התכנות מוכוון-עצמים
43	פרק 2: מהו .NET של Microsoft
55	פרק 3: מתחילים
<b>75</b>	<b>חלק 2 - יסודות המחלקה של C#</b>
77	פרק 4: CTS - Common Type System
89	פרק 5: מחלקות - Classes
123	פרק 6: שיטות
143	פרק 7: מאפיינים (Properties), מערכים (Arrays) וסדרנים (Indexers)
163	פרק 8: תכונות - Attributes
181	פרק 9: ממשקים - Interfaces
<b>207</b>	<b>חלק 3 - כתיבת קוד</b>
209	פרק 10: ביטויים ואופרטורים
235	פרק 11: בקרת זרימה
265	פרק 12: טיפול בשגיאות באמצעות חריגים
	פרק 13: העמסת אופרטורים והמרות מותאמות -
287	Operator Overloading and User-Defined Conversions
301	פרק 14: נציגים ומטפלים באירועים - Delegates and Event Handlers
<b>321</b>	<b>חלק 4 - נושאים מתקדמים ב-C#</b>
323	פרק 15: תכנות עם ריבוי מטלות - Multithreaded Programming
347	פרק 16: Reflection
365	פרק 17: Unmanaged Code
389	פרק 18: Assemblies
<del>405</del>	<del>נספח: התקליטור המצורף</del>
413	Index

# תוכן עניינים

13 ..... מבוא

17 ..... הקדמה

## **21 ..... חלק 1 - הנחת היסודות**

**23 ..... פרק 1: יסודות התכנות מוכוון-עצמים**

25 ..... כל דבר הוא אובייקט

29 ..... אובייקטים לעומת מחלקות

29 ..... instantiation

31 ..... שלושת העיקרים של שפות תכנות מוכוונות-עצמים

31 ..... encapsulation

34 ..... Inheritance

37 ..... Polymorphism

41 ..... סיכום

**43 ..... פרק 2: מהו .NET של Microsoft**

43 ..... פלטפורמת .NET של Microsoft

44 ..... סביבת העבודה .NET

44 ..... .NET ו-Windows DNA

45 ..... Common Language Runtime

46 ..... ספריות המחלקה בסביבת העבודה .NET

48 ..... MSIL ו-JITers

50 ..... Common Type System

50 ..... Reflection ו-Metadata

51 ..... אבטחה

52 ..... Deployment

52 ..... Unmanaged Code

53 ..... סיכום

## פרק 3: מתחילים ..... 55

55	.....	כתיבת יישום C# הראשון
55	.....	בחירת עורך
58	.....	"Hello, World"
58	.....	השימוש במהדר שורת הפקודה
59	.....	הפעלת היישום
60	.....	סקירת הקוד
60	.....	תכנות במקום אחד
61	.....	Members ו- Classes
61	.....	השיטה Main
62	.....	השיטה System.Console.WriteLine
62	.....	Namespaces ו- using Directive
63	.....	Skeleton code
64	.....	אי-בהירות של מחלקות
65	.....	שגיאות ופתרון
65	.....	Compile-Time Errors
66	.....	ILDASM
67	.....	"Hello, World" ב-MSIL
70	.....	קווים מנחים לתכנות ב- C#
70	.....	מתי להגדיר מרחבי שמות משלך
70	.....	קווים מנחים לבחירת שמות
71	.....	תקני מוסכמות לקביעת שמות
74	.....	סיכום

## חלק 2 - יסודות המחלקה של C# ..... 75

### פרק 4: CTS - Common Type System ..... 77

77	.....	כל דבר הוא אובייקט
78	.....	Value Types ו- Reference Types
78	.....	Value Types
79	.....	Reference Types
79	.....	Boxing ו- Unboxing
80	.....	המקור לכל הסוגים : System.Object
81	.....	Aliases ו- Types
82	.....	Casting Between Types
85	.....	Namespaces
85	.....	מילת המפתח using
86	.....	יתרונות CTS
86	.....	תפעוליות שיתופית בין שפות
87	.....	היררכיה המבוססת על מחלקה בודדת
87	.....	Type Safety
88	.....	סיכום

## פרק 5: מחלקות - Classes 89

89	הגדרת מחלקות
90	Class Members
91	Access Modifiers
92	השיטה Main
93	ארגומנטים של שורת-פקודה
94	ערכים מוחזרים
94	הרבה מחלקות עם השיטה Main
95	Constructors
97	Static Members ו- Instance Members
99	Constructor Initializers
102	Constants vs. Read-Only Fields
102	Constants
103	Read-Only Fields
105	Garbage Collector
106	מעט היסטוריה
107	Deterministic Finalization
108	ביצועים
114	הפתרון המושלם
115	פתרון שהוא (כמעט) מושלם
116	Dispose Design Pattern
117	Inheritance
119	Multiple Interfaces
120	Sealed Classes
121	סיכום

## פרק 6: שיטות 123

123	הפרמטרים ref ו-out
128	Method Overloading
131	שיטה עם מספר לא ידוע של פרמטרים
132	Virtual Methods
133	Method Overriding
134	Polimorphism
140	Static Method
142	גישה לחברי מחלקה
142	סיכום



## פרק 7: מאפיינים (Properties), מערכים (Arrays)

143	וסדרנים (Indexers)
143	Properties as Smart Fields
144	הגדרת מאפיינים ושימוש בהם
146	מה באמת עושה כאן המהדר
148	מאפיינים לקריאה בלבד
149	הורשת מאפיינים
149	שימוש מתקדם במאפיינים
150	מערכים
150	הצהרה על מערכים
151	דוגמה של מערך חד-מימדי
152	מערכים רב-מימדיים
154	אחזור מספר המימדים
155	מערכים משוננים
157	Indexers
158	הגדרת סדרנים
158	דוגמה לסדרן
160	קווים מנחים לתכנון
161	סיכום

## פרק 8: תכונות - Attributes

164	הצגת תכונות
165	הגדרת תכונות
166	קריאת מידע של תכונות
166	תכונות של מחלקות
169	תכונות של שיטות
171	תכונות של שדות
173	פרמטרים של תכונות
173	פרמטרים לפי מיקום ופרמטרים לפי שם
175	שגיאות נפוצות עם פרמטרים לפי שם
175	סוגים חוקיים של פרמטרים עבור תכונות
176	AttributeUsage התכונה
176	הגדרת יעד התכונה
178	תכונות לשימוש יחיד ולשימוש מרובה
179	הגדרת חוקי הורשה לתכונה
179	Attribute Identifiers
180	סיכום

<b>181</b>	<b>פרק 9: ממשקים - Interfaces</b>
182	השימוש בממשק
183	הצהרה על ממשקים
184	מימוש ממשקים
186	אחזור מידע אודות יישום ממשקים באמצעות is
190	אחזור מידע אודות מימוש ממשקים באמצעות as
193	שיוך מפורש של שם חבר ממשק
193	הסתרת שמות באמצעות ממשקים
196	מניעת אי-בהירות עם שמות
200	ממשקים והורשה
203	איחוד ממשקים
205	סיכום

## **207 חלק 3 - כתיבת קוד**

<b>209</b>	<b>פרק 10: ביטויים ואופרטורים</b>
209	הגדרת אופרטורים
210	קדימות אופרטורים
211	קביעת קדימות ב- C#
211	אסוציאטיביות ימנית ושמאלית
212	שימוש מעשי
213	אופרטורים של C#
213	אופרטורים בסיסיים של ביטויים
218	אופרטורים מתמטיים
226	אופרטורים של יחסים
229	אופרטורים פשוטים של הצבה
233	סיכום

<b>235</b>	<b>פרק 11: בקרת זרימה</b>
235	משפטי בחירה
235	משפט if
237	ריבוי תנאים במשפט if
238	אכיפת חוקי if ב- C#
240	משפט switch
242	צירוף תוויות case
244	מניעת גלישה במשפטי switch
245	משפטי איטרציה
245	משפט while
247	משפט do/while
249	משפט for
252	משפט foreach

254	הסתעפות עם משפטי דילוג
254	משפט break
255	יציאה מלולאות אינסופיות
257	משפט continue
258	משפט goto
264	סיכום

## פרק 12: טיפול בשגיאות באמצעות חריגים

265	סקירה כללית על טיפול בחריגים
267	תחביר בסיסי של טיפול בחריגים
267	Throwing an Exception
267	Catching an Exception
268	Rethrowing an Exception
270	ניקוי בעזרת finally
270	השוואה בין טכניקות לטיפול בשגיאות
271	היתרון של טיפול בחריגים על פני קודים מוחזרים
273	טיפול בשגיאות בהקשר הנכון
274	שיפור קריאות הקוד
276	Throwing Exceptions From Constructors
276	שימוש במחלקה System.Exception
276	בניית אובייקט Exception
279	שימוש במאפיין StackTrace
280	איך לתפוס סוגי חריגים מרובים
281	גזירת מחלקות Exception מותאמות
283	תכנון קוד עם טיפול בחריגים
283	סוגיות תכנון של בלוק try
285	סוגיות תכנון של בלוק catch
286	סיכום

## פרק 13: העמסת אופרטורים והמרות מותאמות -

### Operator Overloading and

### 287 User-Defined Conversions

287	Operator Overloading
288	תחביר ודוגמה
291	האופרטורים שניתן להעמיס
291	הגבלות על העמסת אופרטורים
292	קווים מנחים לתכנון
292	User-Defined Conversions
293	תחביר ודוגמה
299	סיכום

## פרק 14: נציגים ומטפלים באירועים -

### 301..... Delegates and Event Handlers

301	שימוש בנציגים כשיטות משוב
305	הגדרת נציגים כחברים סטטיים
307	יצירת נציגים על פי צורך
309	הרכבת נציגים
315	הגדרת אירועים באמצעות נציגים
319	סיכום

## חלק 4 - נושאים מתקדמים ב-C# 321.....

### פרק 15: תכנות עם ריבוי מטלות -

### 323..... Multithreaded Programming

323	יסודות השימוש במטלות
324	מטלות וריבוי משימות
324	context switching
325	יישום C# עם ריבוי מטלות
326	עבודה עם מטלות
326	AppDomain
327	המחלקה Thread
330	תזמון מטלות
334	בטיחות וסינכרון מטלות
335	הגנה על קוד באמצעות המחלקה Monitor
339	משפט lock
341	סינכרון קוד באמצעות המחלקה Mutex
343	בטיחות מטלות ומחלקות NET
343	קווים מנחים לשימוש במטלות
343	מתי להשתמש במטלות
344	מתי אין להשתמש במטלות
345	סיכום

### פרק 16: Reflection 347.....

347	היררכיית API של השיקוף
348	המחלקה Type
348	קבלת אובייקט Type משם משתנה
349	קבלת אובייקט Type משם של סוג
349	Interrogating Types
352	Modules ו-Assemblies
352	ביצוע איטרציה על הסוגים של מכלל קוד
355	פירוט המודולים של מכלל קוד

357	.....	Late Binding with Reflection
360	.....	יצירה וביצוע קוד בזמן ריצה
363	.....	סיכום

## 365..... פרק 17: Unmanaged Code

366	.....	שירותי הפעלת פלטפורמות
366	.....	הצהרה על פונקציית DLL חיצונית
369	.....	Callbacks ו- PInvoke
370	.....	Marshalling ו- PInvoke
371	.....	Unsafe Code
372	.....	Pointers
373	.....	המשפט fixed
375	.....	COM
375	.....	עולם חדש ונועז
376	.....	מתחילים
377	.....	Metadata ו- Com typelib
380	.....	Early Binding to COM Components
382	.....	Query Interface
383	.....	Late Binding to COM Components
385	.....	COM Threading Models
386	.....	סיכום

## 389..... פרק 18: Assemblies

389	.....	סקירה כללית
390	.....	Manifest Data
391	.....	Assembly Packing
391	.....	Assembly Deployment
392	.....	Assembly Versioning
392	.....	Building Assemblies
393	.....	Multiple Modules
395	.....	Creating Shared Assemblies
397	.....	Global Assembly Cache
397	.....	עיון במטמון
399	.....	Versioning Assemblies
402	.....	QFEs and the Default Version Policy
402	.....	Safe Mode Configuration File
404	.....	סיכום

## ~~405..... נספח: התקליטור המצורף~~

## 413..... Index

## C# למתכנתי JAVA/C++

מאת **Scott Wiltamuth**, חבר בצוות  
הפיתוח של C# בחברת Microsoft.

כל שנות עבודתי במיקרוסופט הושקעו בניסיון לשפר את יכולתו של מפתח התוכנה, ובעיקר להגדיל את יצרנותו. עבודתי הקיפה מיגוון רחב של מוצרים וטכנולוגיות, אך מעולם לא הייתי כה נלהב מעבודתי כפי שאני עכשיו. הטכנולוגיות שמספקת Microsoft .NET מדהימים ברוחבם ובעומקם. אנו מספקים שפה חדשה ונהדרת, השוברת את כל המחסומים אשר באופן מסורתי חילקו את המפתחים לעולמות שונים ובלתי-שקולים, ומאפשרת שיתוף פעולה בין אתרי אינטרנט כדי לענות על צרכי המשתמשים.

הבה נסקור את אבני היסוד של NET ומספר טכנולוגיות קשורות:

- ❖ שפת C#, שפה חדשה – C# היא השפה הראשונה מוכוונת הרכיבים (component oriented) במשפחת השפות C ו-C++. שפה זו, הנגזרת מ-C ו-C++, פשוטה, מודרנית, מוכוונת עצמים ומאפשרת לעבוד עם סוגי נתונים מורשים בלבד. C# משלבת את היצרנות הגבוהה של Microsoft Visual Basic עם העוצמה הגלומה ב-C++.
- ❖ **Common Language Runtime** – סביבת CLR עם ביצועים מעולים, הכוללת מנוע ביצוע, garbage collector (אוסף האשפה), הידור במועד Just-In-Time, מערכת אבטחה ו-NET Framework.
- ❖ **Common Language Specification** – תקן CLS מתאר רמה משותפת של פונקציונליות עבור שפת תכנות. החסם המינימלי יחסית של CLS מאפשר יצירת "מועדון" של שפות תואמות – CLS. כל אחד מחברי המועדון נהנה משני יתרונות: גישה מלאה לפונקציונליות של מסגרת NET ותפעוליות שיתופית עשירה עם שפות תואמות אחרות. לדוגמה, מחלקה של Visual Basic יכולה לרשת ממחלקה של C# ולדרוס את השיטות הווירטואליות שלה.
- ❖ **אוסף שפות עשיר המיועד ל-CLR** – מיקרוסופט מספקת שפות העומדות בדרישות CLS, כמו Visual Basic, Visual C++, עם הרחבות, Visual C# ו-JScript. בנוסף לאלו קיימים שפות צד ג' אחרות רבות מכדי לפרטן כאן.

❖ **שירותי אינטרנט (Web)** – רשת האינטרנט מורכבת כיום בעיקר מאתרים נפרדים. כאשר משתמש מבקר באתרים מרובים כדי לבצע משימה נתונה, כגון סידורי נסיעה עבור קבוצת אנשים, אתרים אלה אינם משתפים פעולה ביניהם. הדור הבא של האתרים יתבסס על שיתוף רשתות של אתרי האינטרנט. הסיבה לכך פשוטה: שיתופיות בין אתרי אינטרנט יכולה לפעול טוב יותר כדי לענות על צרכי המשתמשים. טכנולוגיות שירותי אינטרנט מעודדות שיתוף פעולה בין אתרים בכך שהן מאפשרות תקשורת באמצעות פרוטוקולים מבוססי-XML סטנדרטיים שהם בלתי-תלויים בשפה ובלתי-תלויים בפלטפורמה גם יחד. שירותי אינטרנט רבים יתבססו על C# ועל CLR המתבצעים על Windows, אך אפשרויות הארכיטקטורה אינן מוגבלות.

❖ **Visual Studio .NET** – קושר יחד את כל המרכיבים ומאפשר ליצור בקלות מיגוון רחב של רכיבים, יישומים ושירותים, בשפות תכנות שונות.

לאחר שנגעתי במספר טכנולוגיות חשובות הקשורות ל-C#, הבה נסקור את C# עצמה. מפתחים השקיעו ומשקיעים רבות בשפה שבה בחרו, ולכן מחובתנו להוכיח את כדאיותה של שפה חדשה. נעשה זאת באמצעות שילוב מסוים בין שימור פשוט, שיפורים מתקנים וחדשנות מעמיקה.

## שימור פשוט

הפילוסוף היווני היפוקרטס אמר "עשה לך הרגל של שני דברים: לעזור, או לפחות לא להזיק". החלק של "לפחות לא להזיק" שיחק תפקיד חשוב בתכנון של C#. אם C או C++ טיפלו בבעיה היטב, יכולת זו נשמרה ללא שינוי. חשוב מכך, C# מעתיקה מ-C ו-C++ ביטויים, משפטים וגם את התחביר הכללי. מכיון שחלק נכבד של כל תוכנית טיפוסית מכיל את כל התכונות האלו, מתכנתי C ו-C++ לא יתקשו בעבודה עם C#.

## שיפורים מתקנים

ב-C# נעשו שיפורים רבים – יותר מדי מכדי להזכירם בהקדמה קצרה זו – אך מן הראוי לציין מספר שינויים שבאים למנוע שגיאות רגילות ושגיאות זוללות זמן של C ו-C++:

❖ חובה להכריז על משתנים לפני שמשתמשים בהם, כדי למנוע שגיאות הנגרמות ממשתנים לא מאותחלים.

❖ התנאים במשפטים כמו if ו-while דורשים ערכים בוליאניים, כך שמפתח המשתמש בטעות באופרטור ההצבה (=) במקום באופרטור השוואה (==) מאתר את השגיאה בזמן ההידור.

❖ כישלון שקט (silent fall-through) במשפטי switch נמנע; כך, מתכנת המשמיט בטעות משפט break יכול לאתר את השגיאה בזמן ההידור.

## חדשנות מעמיקה

חדשנות מעמיקה נמצאת במערכת הסוגים המשותפים (CTS - Common Type System) של C#, וכוללת את החידושים הבאים:

- ❖ מערכת הסוגים של C# מיישמת ניהול זיכרון אוטומטי, ובכך משחררת את המפתחים מניהול ידני צורך-זמן ומועד לשגיאות. שלא כמרבית מערכות הסוגים האחרות, מערכת הסוגים של C# גם מאפשרת מניפולציה על סוגי מצביעים ועל כתובות של אובייקטים (טכניקות ניהול זיכרון ידניות אלו מורשות רק בהקשרי אבטחה מסוימים).
- ❖ מערכת הסוגים של C# הינה עקבית – כל דבר הוא אובייקט. C# מגשרת על הפער בין סוגי ערכים (value types) וסוגי ייחוס (reference types) באמצעות שימוש במושגים מרכזיים וחדשניים הנקראים boxing (אריזה) ו-unboxing (פריקה). כך היא מאפשרת להתייחס אל כל פיסת נתונים כאל אובייקט.
- ❖ מאפיינים (properties), שיטות (methods) ואירועים (events) הם מרכיבים חיוניים של השפה. שפות רבות אחרות חסרות תמיכה פנימית עבור מאפיינים ואירועים, ובכך יוצרות חוסר התאמה מיותרת בין השפה לבין המערכות. לדוגמה, אם המערכת תומכת במאפיינים והשפה אינה עושה זאת, הגדלת ערך המאפיין הופכת להיות מסורבלת, לדוגמה, `o.SetValue(o.GetValue()+1)`. אם גם השפה תומכת במאפיינים, הפעולה היא פשוטה, `o.Value++`.
- ❖ C# תומכת בתכונות (attributes) המאפשרות להגדיר ולהשתמש במידע הצהרתי על רכיבים. היכולת להגדיר סוגים חדשים של מידע הצהרתי היתה תמיד כלי רב-עוצמה עבור מתכנני השפה. יכולת זו נמצאת כעת ברשותם של כל מפתחי C#.

## על הספר

המחבר Tom Archer בונה את תשתית הלימוד על ידי הצגת NET ו-CLR. הוא מסביר את מרכיבי היסוד של C# ויורד לעומקם של מספר מושגים מתקדמים בשפה זו. ניסיונו הרב – הן כמפתח והן כמחבר ספרים על C++, J++ ו-Microsoft Windows – מאפשר לו להסביר את C# בדרכים מהנות ומאלפות.

קורא יקר, אני מקווה שתיהנה לכתוב את תוכניות C# הראשונות שלך, שתהיינה הראשונות בתוכניות רבות שתכתוב בעתיד.

**Scott Wiltamuth**

C# Design Team Member

Microsoft Corporation

### הערה מפי העורך:

Scott Wiltamuth, עובד Microsoft, טוען (ויש צדק בטענותיו) שמתכנתי C/C++ ירגישו בנוח בעבודה עם C#. לדעתי (הצנועה) מתכנתי Java ירגישו נוח יותר. לא לחינם קוראים ל-Microsoft C# בשם Microsoft Java Killer.



## מדוע כתבתי את הספר

כמפתח תוכנה במשך עשרים שנה – אני חש מבוגר יותר בכל פעם שאני חושב על כך – הגעתי לנקודה שבה תכנות החל להיות מעט נדוש עבורי. אל תבינו אותי לא נכון: אם הייתי מולטי מיליונר ולא הייתי צריך לעבוד, קרוב לוודאי שהייתי ממשיך בכתיבת קוד, כי זה דבר שבאמת אני נהנה לעשותו. עם זאת, הגעתי לנקודה שבה חשבתי לעצמי, "הכל כבר נעשה!". אז הופיעו Microsoft .NET ו-C#, ועולם חדש ושלם של אפשרויות נפתח בפניי. שוחחתי עם מספר חברים שחוו את אותה תחושה של "התעוררות מחדש" עם הצגת NET לראשונה. יש בידינו את הטכנולוגיה החדשה והמלהיבה הזאת, אשר סוף סוף מתמודדת עם סוגיות שטיפלנו בהן במשך שנים (לדוגמה, סביבות פיתוח מרובות שפות, בעיות פיתוח וניהול גרסאות של מערכות גדולות ומורכבות, וכן הלאה). כתבתי ספר זה כי החזרה לכתיבת קוד מרתקת אותי, ושוב אני מתעורר בכל בוקר וחושב על הדבר החדש והמלהיב שאלמד היום. אני מקווה כי במהלך לימוד השפה תהיה גם אתה שותף להתלהבות זו.

נכון לזמן כתיבת הספר, כל הכותב ספר על C# לומד למעשה את השפה בזמן הכתיבה. אם כתבת יישום במהלך לימוד SDK או C#, אתה יודע שזו עשויה להיות חוויה לא קלה. כעת נסה לדמיין שאתה כותב משהו שעשרות אלפי אנשים עומדים לקרוא לאחר שסיימת! הבעיה הגדולה היא שבמחצית הדרך – ברגע שהבנת מה אתה עושה – אתה מתחבט בתכנון וכתיבה מחדש של הכל! כמובן שאין זה מעשי כאשר קיימים תאריכי יעד סופיים. אני מאמין שספר זה משמש כלי טוב ללימוד C#. היות ולמדתי תוך כדי כתיבתו, עשויה להיות מידה מסוימת של חוסר עקביות ועשויים להיות מספר דברים שניתן היה לשפר אותם בהרבה. עם זאת, אם תהיה לי הזדמנות לכתוב מהדורה שנייה, אני יכול להבטיח שגם אתה תוכל להפיק תועלת מעקומת הלמידה האישית שלי.

## שגיאות

אני יצור אנושי שלפעמים עושה טעויות. אני מכיר בזה ומוכן לקבל בברכה כל משוב בנוגע לספר שכתבתי. איני מאלה האומרים "אני כל כך טוב כי אני כותב ספרים". אני רק בחור רגיל שהתמזל מזלו וקיבל הזדמנות לכתוב ספר. אני תמיד פתוח ללמוד מאחרים, ולמעשה אוהב לעשות כך. ניתן להגיע אלי בכתובת <http://www.TheCodeChannel.com>. באתר תוכלו למצוא תחת הכותרת Errata את רשימת הדברים שיש לתקן בספר.

**הערות העורך:** במהדורה העברית **תוקנו** כל ההערות הידועות נכון ליום הוצאת הספר לאור. לא כן במהדורה האלקטרונית המלאה של הספר באנגלית המצורף לתקליטור. בכל מקרה, אני ממליץ לכל אחד להתעדכן בדף Errata באתר המחבר.

## למי מיועד הספר

ספר זה מיועד למי שרוצה להתחיל ללמוד פיתוח C# ו-.NET. כפי שצינתי, זו פלטפורמה מרתקת עם עתיד מבטיח בכל הנוגע לפיתוח מבוסס עבור Microsoft Windows. ספר זה מניח שיש לך רקע באחת מהשפות C, ++C, ++C Visual או Java. אני סבור שהתנאי המוקדם הנוסף הוא רק הרצון ללמוד ולגלות מימדים חדשים בכתובת יישומים. מכיון שאתה אוהז בספר, כנראה שיש לך את הרצון הזה.

## מבנה הספר

ספר זה אורגן בקפידה כרצף לוגי של מספר חלקים; כל חלק מכיל קבוצת פרקים שעוסקים בנושא מרכזי של פיתוח C# או .NET.

הספר מתחיל בחלק 1 – "הנחת היסודות", המיועד למתכנתי C# מתחילים ולא לה שסביבת NET חדשה עבורם. הפרקים בחלק זה מספקים הקדמה ל-OOP וסביבת NET ומדגימים איך ליצור ולבדוק את תוכניות C# הראשונות.

חלק 2 – "יסודות המחלקה של C#", מציג את היסודות של הגדרה ועבודה עם מחלקות (classes) ב-C#. הפרקים בחלק זה מכוונים לספק לך בסיס מוצק אודות החברים הנתמכים ב-C# (enums, properties, arrays וכן הלאה) ואיך להגדירם ולהשתמש בהם ביישום C#.

למרות שבפרקים קודמים כתבת קוד במונחים של הגדרת חברי מחלקה (class members), בחלק 3 – "כתיבת קוד", תתחיל לבחון היבטים אחרים של משימות כמו זרימת תוכנית (program flow), טיפול בשגיאות עם חריגים (exception) וכתובת מטפלים באירועים (event handlers) באמצעות נציגים (delegates).

חלק 4 – "נושאים מתקדמים ב-C#", מסיים את הספר. בהיותי "מכור מחשבים" טיפוסי, נהייתי מאד לכתוב אותו. חלק זה מכיל פרקים על תכנות מרובה מטלות (multithreading), שיקוף (reflection), עבודה עם קוד לא-מנוהל (unmanaged code) הכולל תפעוליות שיתופית עם רכיבי COM, והבחנת גרסאות (versioning).

## אודות התקליטור המצורף

לספר זה מצורף תקליטור. ראה נספח בסוף הספר – התקליטור המצורף – להוראות התקנה והפעלה של התקליטור.

אם במחשב שלך מסומנת האפשרות AutoRun (הפעלה אוטומטית של תקליטור) הקיימת ב-Windows, יופיע חלון פתיחה כאשר תכניס את התקליטור לכוון התקליטורים ויוצגו מספר אפשרויות התקנה. בכדי לפתוח חלון זה באופן ידני הפעל את התוכנית **Hodami.exe** מספריית השורש של התקליטור.

### ספר אלקטרוני eBook

את הגירסה המלאה האלקטרונית של הספר באנגלית תוכל למצוא בתקליטור, בתיקה **59327\ eBook**. אם ביצעת התקנה לדיסק במחשב שלך, תוכל למצוא אותה באותה תיקיה. אתר והפעל את קובץ **setup.exe**.

### קוד מקור

תוכניות הדוגמה של הספר נמצאות בתקליטור המצורף, בתיקה **59327**. תוכל לעיין בדוגמאות ישירות מהתקליטור, או להתקין בדיסק באמצעות תוכנית ההתקנה הניתנת להפעלה דרך **Hodami.exe**.

### דרישות מערכת

כדי להפיק את המירב מספר זה, מומלץ מאוד לעיין ביישומי הדוגמה במהלך הקריאה של כל פרק. לפיכך יש להתקין את הגירסה האחרונה של .NET Framework SDK. בזמן כתיבת שורות אלו, ערכת הפיתוח כוללת את סביבת הריצה של NET ואת מהדר C#. נוסף לכך, נמנעתי במתכוון להשתמש במוצר Visual Studio .NET בכדי להתמקד בשפה ובסביבת הריצה, ולא להציב מגבלות כלשהן על סביבת הפיתוח המסוימת שלך. משום כך, כל הדוגמאות בספר ניתנות להידור וביצוע משורת הפקודה.

### על היועץ המקצועי

**ארז קורן**, בוגר יחידת המחשב בצה"ל – ממר"מ, מתמחה בעולם האינטרנט, e-Commerce וטכנולוגיות חדשניות. בעל ניסיון עשיר בפיתוח ביישומי C++ ו-Java ליישומים עתירי פלטפורמות ומשתמשים. כיום, מנהל צוות פיתוח בחברת Comverse בתחום המתפתח של המסחר האלקטרוני.



חלק 1

# הנחת היסודות





# פרק 1

## יסודות התכנות מוכוון-עצמים

פרק זה יצעיד אותך אל עולם המושגים של **תכנות מוכוון-עצמים – OOP** (Object-Oriented Programming), או **מוכוון אובייקטים**, ויעזור לך להבין את החשיבות של עקרונות אלה לתכנות. על שפות תכנות רבות, כגון ++C או Visual Basic, נאמר שהן "תומכות באובייקטים", אך למעשה רק שפות מעטות תומכות בכל העקרונות שתכנות מוכוון-עצמים מושתת עליהם. #C היא אחת מהשפות האלו: היא תוכננה מיסודה להיות שפה מכוונת-עצמים ומבוססת-רכיבים אמיתית. לכן, כדי להפיק את מלוא התועלת מספר זה, עליך להבין היטב את העקרונות שיוצגו כאן.

קוראים המבקשים "לצלול" ישר לתוך הקוד, נוהגים בדרך כלל לדלג על פרקים העוסקים במושגים ועקרונות. עם זאת, מומלץ שתקרא פרק זה בעיון, אלא אם אתה מומחה בנושא. גם אם יש לך מושג כלשהו בתכנות מוכוון-עצמים, תוכל להפיק תועלת רבה מקריאת הפרק. זכור שהפרקים הבאים יתייחסו למונחים ולעקרונות הנידונים כאן, ולכן בכל מקרה כדאי לסקור את תוכנו.

שפות רבות אמורות להיות מוכוונות-עצמים או מבוססות-עצמים (אובייקטים), אך רק מעטות הן כאלו באמת. שפת ++C אינה כזו, כי שורשיה נעוצים בשפת C, עובדה שלא ניתן לשנות. אידיאלים רבים מאוד של תכנות מוכוון-עצמים הוקרבו בשפת ++C לטובת תמיכתה בקוד הקיים שנכתב בשפת C. אפילו לשפת Java, טובה ככל שתהיה, יש מספר מגבלות כשפה מוכוונת-עצמים. ליתר דיוק, אתיחס לעובדה שבשפה זו קיימים סוגי אובייקטים וסוגי נתונים בסיסיים המטופלים ומתנהגים באופן שונה מאוד זה מזה. עם זאת, פרק זה אינו מתמקד בהשוואת הנאמנות של שפות שונות לעקרונות של OOP, אלא מציג לימוד אובייקטיבי של עקרונות אלה.

לפני שנתחיל, ברצוני להוסיף שתכנות מוכוון-עצמים הוא הרבה יותר מצירוף מילים מסחרי (למרות שהפך לכזה עבור מספר אנשים), הוא יותר מתחביר חדש, או מממשק תכנות יישומים (API) חדש. תכנות מוכוון-עצמים הוא מערכת שלמה של עקרונות ורעיונות, המייצגים דרך חשיבה לטיפול ולהתמודדות אינטואיטיביים בתוכנית מחשב, ולפיכך זהו תכנות יעיל יותר.

עבודתי הראשונה היתה כרוכה בכתיבת יישום מסחרי בשפת Pascal. בהמשך כתבתי יישומים ב-PL/I וב-RPG III (וגם ב-RPG/400). כעבור מספר שנים התחלתי לכתוב יישומים בשפת C. בכל אחד מהמקרים יכולתי ליישם בקלות את הידע שצברתי מהניסיון הקודם. עקומת הלמידה התקצרה במעבר משפה לשפה ללא תלות במורכבות השפה שלמדתי. הסיבה לכך היא שעד שהתחלתי לכתוב תוכניות ב-C++, כל השפות שבהן השתמשתי היו שפות פרוצדורליות הנבדלות זו מזו בעיקר בתחביר.

אם תכנות מוכוון-עצמים חדש לך, עלי להזהירך: ניסיון קודם עם שפות שאינן מוכוונות-עצמים לא יועיל לך כאן! תכנות מוכוון-עצמים הוא דרך חשיבה שונה על אופן התכנון והתכנות של פתרונות לבעיות. למעשה, מחקרים מוכיחים שמתכנתים מתחילים לומדים שפות מוכוונות-עצמים הרבה יותר מהר מאלה אשר החלו בשפות פרוצדורליות כמו BASIC, COBOL ו-C. כל אלה אינם צריכים "להשתחרר" מהרגלים פרוצדורליים ולשנות דפוסי חשיבה שעלולים להקשות על הבנת OOP, אלא מתחילים מהבסיס. אם כתבת תוכניות בשפות פרוצדורליות במשך שנים רבות ו-C# היא השפה מוכוונת-העצמים הראשונה שלך, העצה הטובה ביותר שאוכל לתת לך היא לשמור על ראש פתוח כדי ליישם את הרעיונות שאציג כאן בטרם תיכנע לטובת פתרון לא אמיתי לבעיה בשפה פרוצדורלית כלשהי. כל מי שמגיע לתכנות מוכוון-עצמים מרקע של שפה פרוצדורלית עובר תהליך המתבטא בעקומת למידה, ואשר משתלם לו בסיום התהליך. התועלת בכתיבת תוכניות בשפה מוכוונת-עצמים רבה מאוד, הן במונחים של תכנות יעיל יותר והן בבניית מערכת שניתן לשנותה ולהרחיבה בקלות מרגע שנכתבה. אולי בהתחלה אין זה נראה כך, אך במהלך כ-20 שנה של פיתוח תוכנה (כולל 8 האחרונות בשפות מוכוונות-עצמים) למדתי שעקרונות OOP אכן עונים על הציפיות מהם, **אם הם מיושמים באופן נכון**.



# כל דבר הוא אובייקט

בשפה מוכוונת עצמים אמיתית, כל היישויות של **מרחב הבעיה** (problem domain) מבוטאות באמצעות הרעיון של **אובייקטים** (שים לב שבספר זה אשתמש בהגדרה של Coad/Yourdon ל"מרחב הבעיה" – שמתאר את הבעיה שמנסים לפתור, במונחים המייחדים אותה, כמו מורכבות, מושגים, אתגרים וכו'). כפי שאולי ניחשת, השימוש ב**אובייקטים** (objects) הוא הרעיון המרכזי שעומד מאחורי תכנות מוכוון-עצמים. בדרך כלל אין אנו חושבים במונחים של מבנים, חבילות נתונים, קריאות לפונקציות ומצביעים, אלא במונחים של אובייקטים. הנה דוגמה הממחישה זאת.

אם היית כותב יישום להפקת חשבוניות והיית צריך לסכם את שורות הפירוט של החשבונית, איזו גישה מחשבה מבין הבאות היתה אינטואיטיבית יותר מנקודת המבט של הלקוח?

❖ **גישה שאינה מוכוונת-עצמים** - תהיה לי גישה למבנה נתונים המייצג את כותרת החשבונית. מבנה כותרת החשבונית יכלול גם רשימה מקושרת של מבנים המייצגים את פירוט החשבונית, כאשר כל מבנה מכיל ערך שורת סכום. לכן, כדי לקבל את הסכום הכולל של החשבונית, עלי להצהיר על משתנה הנקרא למשל `totalInvoiceAmount`, לאתחל אותו בערך 0, לקבל מצביע למבנה כותרת החשבונית, לקבל את ראש הרשימה המקושרת של שורות הפירוט, ואז לנוע על פני הרשימה. בכל קריאה של מבנה שורת פירוט, אקבל את **משתנה המחלקה** (member variable) שלו המכיל את הסיכום עבור אותה שורה, ואוסיף אותו למשתנה `totalInvoiceAmount`.

❖ **גישה מוכוונת-עצמים** - יהיה לי אובייקט חשבונית, ואשלח לו הודעה המבקשת ממנו את הסכום הכולל של החשבונית. איני צריך לחשוב על האופן שבו האובייקט שומר את המידע הפנימי שלו, כפי שהיה עלי לעשות עם מבנה הנתונים בגישה הקודמת. אני פשוט נוהג עם האובייקט בדרך טבעית של הפניית בקשות באמצעות שליחת הודעות (אוסף הבקשות שאותן האובייקט מסוגל לעבד נקרא בשם כללי **ממשק האובייקט** (object's interface). בהמשך אסביר מדוע חשיבה במונחים של **ממשק** (interface) ולא של **מימוש** (implementation), או יישום, מוצדקים בגישה מוכוונת-עצמים, כפי שעשיתי כאן).

אין ספק שגישה מוכוונת-עצמים היא אינטואיטיבית וקרובה יותר לדרך החשיבה של רוב האנשים המטפלים בבעיה העומדת בפניהם. בפתרון השני (מוכוון-עצמים), סביר שאובייקט החשבונית מבצע איטרציה על **אוסף** (collection) אובייקטים של פרטי חשבונית ושולח לכל אחד מהם הודעה המבקשת לקבל את שורת הסיכום שלו.

אולם, אם אתה מבקש סיכום כולל, **אינך צריך לטפל באופן הביצוע של התהליך**. אינך דואג לכך, כי אחד מעיקרי תכנות מוכוון-עצמים הוא **כימוס** (encapsulation) – היכולת של אובייקט להסתיר את השיטות והנתונים הפנימיים שלו ולהציג ממשק המאפשר לתוכנית לגשת רק לחלקיו החשובים. אין חשיבות לדרך הפנימית שבה האובייקט מבצע את עבודתו, כל זמן שהוא מסוגל לבצע עבודה זו. הוא מציג ממשק שבאמצעותו אתה גורם לו לבצע עבודה מוגדרת ומועילה עבורך (בהמשך הפרק אסביר ביתר פירוט את העקרונות של כימוס וממשקים). הנקודה החשובה היא, שקל מאוד לתכנן ולכתוב תוכניות מחשב המחקות את האובייקטים של מרחב הבעיה בעולם הממשי, כי הן מאפשרות לנו לחשוב בדרך טבעית יותר.

שים לב שגישת העיבוד מוכוון-עצמים דורשת אובייקט שיבצע עבודה מועילה עבורך – כלומר, סיכום של כל שורות הפירוט. שים לב שאובייקט אינו מורכב מנתונים בלבד, כמו שקיים במבנה נתונים. על פי ההגדרה, אובייקטים כוללים נתונים ושיטות הפועלות על נתונים אלה. כלומר, כאשר מתייחסים למרחב בעיה, ניתן לעשות יותר מאשר תכנון של מבני נתונים הדרושים לבעיה. ניתן גם לבחון אילו שיטות יש לשייך לאובייקט נתון ובכך ליצור יחידה פונקציונלית עם כימוס מלא. הדוגמאות המובאות כאן וגם בחלקים הבאים עוזרות להבהיר עיקרון זה.

קטעי הקוד בפרק זה מציגים את העקרונות של תכנות מוכוון-עצמים. יש לזכור שקטעי קוד רבים אמנם כתובים ב-C#, אך העקרונות עצמם ישימים לכל שפה המוגדרת כ-OOP, ואינם ייחודיים לשפה מסוימת. למטרות השוואה בפרק זה, אציג דוגמאות גם בשפת C, אשר אינה מוכוונת-עצמים.



נניח שאתה כותב יישום המחשב את השכר של העובדת היחידה בשם Amy בחברתך. כדי לשייך נתונים מסוימים לעובדת זו, תכתוב בשפת C קוד, אשר קרוב לוודאי יהיה דומה לקוד הבא:

```
struct EMPLOYEE
{
    char szFirstName[25];
    char szLastName[25];
    int iAge; double
    dPayRate;
};
```

כך תחשב את השכר של Amy, על ידי שימוש במבנה EMPLOYEE :

```
void main()
{
    double dTotalPay;

    struct EMPLOYEE* pEmp;
    pEmp = (struct EMPLOYEE*)malloc(sizeof(struct EMPLOYEE));

    if (pEmp)
    {
        pEmp->dPayRate = 100;

        strcpy(pEmp->szFirstName, "Amy");
        strcpy(pEmp->szLastName, "Anderson");
        pEmp->iAge = 28;

        dTotalPay = pEmp->dPayRate * 40;
        printf("Total Payment for %s %s is %0.2f",
            pEmp->szFirstName, pEmp->szLastName, dTotalPay);
    }
    free(pEmp);
}
```

בדוגמה זו, הקוד מבוסס על נתונים הכלולים במבנה, וקוד כלשהו חיצוני (למבנה) המשתמש במבנה זה. ובכן, מה הבעיה? עיקר הבעיה הוא בהפשטה: המשתמש במבנה EMPLOYEE חייב לדעת הרבה מאוד על אודות הנתונים הנחוצים לעובד. מדוע? ניח שבזמן מאוחר יותר תרצה לשנות את הדרך לחישוב השכר של Amy. לדוגמה, אולי תרצה לכלול מרכיבי מס בתחשיב השכר נטו. יהיה עליך לשנות את כל הקוד המשתמש במבנה EMPLOYEE, וכן יהיה עליך לתעד את עובדת השינוי, לידיעת מתכנתים אחרים.

כעת נתבונן על גרסת C# של דוגמה זו :

```
using System;

class Employee
{
    public Employee(string firstName, string lastName, int age, double payRate)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.payRate = payRate;
    }
}
```

```

protected string firstName;
protected string lastName;
protected int age;
protected double payRate;

public double CalculatePay(int hoursWorked)
{
    // Calculate pay here.
    return (payRate * (double)hoursWorked);
}

}

class EmployeeApp
{
    public static void Main()
    {
        Employee emp = new Employee ("Amy", "Anderson", 28, 100);
        Console.WriteLine("\nAmy's pay is $" + emp.CalculatePay(40));
    }
}

```

בגרסת C# של תוכנית הדוגמה EmployeeApp, המשתמש באובייקט יכול לקרוא לשיטה CalculatePay כדי שהאובייקט יחשב את השכר שלו עצמו (של האובייקט). היתרון בגישה זו הוא שהמשתמש אינו צריך לדאוג לדרך החישוב הפנימית של השכר. אם תחליט בעתיד לשנות את דרך חישוב השכר, לא תהיה לכך שום השפעה על הקוד הקיים. רמה זו של הפשטה היא אחת היתרונות הבסיסיים של השימוש באובייקטים.

כמובן שניתן לפשט את קוד הלקוח ב-C על ידי יצירת פונקציה שתאפשר לגשת למבנה EMPLOYEE. עם זאת, העובדה שיש ליצור פונקציה זו בנפרד לגמרי מהמבנה, היא בדיוק הבעיה. בשפה מוכוונת-עצמים כמו C#, הנתונים של האובייקט והשיטות הפועלות על אותם נתונים (הממשק שלו) נמצאים תמיד יחד.

יש לזכור שרק שיטות של אובייקט צריכות לשנות את המשתנים שלו. כפי שניתן לראות בדוגמה הקודמת, כל משתנה מחלקה של Employee מוגדר עם מציין הגישה **protected** (מוגן), חוץ מהשיטה CalculatePay המוגדרת **public** (ציבורית). מציין גישה משמשים לציון רמת הגישה שיש למחלקות נגזרות ולקוד לקוח אל **חבר מחלקה** (class member) נתון. חבר מחלקה מוגן הינו זמין למחלקות נגזרות, אך אינו זמין לקוד לקוח חיצוני. חבר מחלקה ציבורי זמין גם למחלקות נגזרות וגם לקוד לקוח. בפרק 5 נדון ביתר פירוט במציני גישה, אך כעת חשוב לזכור שמציני גישה מאפשרים להגן על חברי מחלקה (או איברי מחלקה) חשובים מפני שימוש לא נכון בהם.