Properties - מאפיינים

תכנון נכון של מחלקה מסתיר את ה- Data Members מפני כל הגורמים החיצוניים למחלקה (Data Hiding).

אולם מה קורה כאשר צריך לשנות באובייקט את אחד מפרטי המידע? מה קורה כאשר אדם עובר דירה? כאשר עובד מקבל העלאה במשכורת?

במקרים אלו ורבים אחרים נזדקק למכניזם מסוים אשר יאפשר לשנות בזמן ריצה (Run time) את הערכים הקיימים ב- Data Members .

מתכנתי  java ו- C++  מכירים את מתודות הגישה (Accessor Methods),

מתודות Accessor Set ו- Get Accessor אשר באמצעותם בוצעו מניפולציות על ה- Data Members.

על פי אותה הגישה המחלקה אמורה להכיל מתודות מיוחדות לטיפול בכל שדה ושדה, לכל שדה תוכננו שתי מתודות:

·          מתודת Set לשינוי ערכי השדה (כתיבה). (setName)

·          מתודת Get אשר מחזירה את תוכן השדה (קריאה). (getName)

אין שום פסול בגישה זו, אולם C# מתקדמת צעד אחד קדימה ומגדירה מנגנון לטיפול נוח ו'נקי' יותר ב – Data Members. מנגנון זה נקרא Properties (מאפיינים).

מנגנון זה עונה הן על הצורך של השימוש במתודות Set והן על הצורך של השימוש במתודות Get.

ניתן להשתמש הן ב- Set והן ב- Get עבור ה- Data Member או רק באחד מאותם Accessors.

Property שמגדיר רק בלוק get מכונה Read Only property ,

Property שמגדיר רק בלוק set מכונה Write only property .

לא חובה להגדיר property עבור כל  ה- Data members.

C# מאפשרת להגדיר גם Static Properties עבור Static Data Members.

Public ReturnType PropertyName

{

get

{

   return data\_member;

       }

      set

      {

  data\_member = value;

       }

}

value הנו משתנה אוטומטי שמוגדר בתוך set  ונועד להשמת המידע החדש בשדה הרלוונטי.

Value  הנה מילה שמורה והטיפוס ממנו נוצר משתנה אוטומטי זה מוגדר ב – Prototype של ה – Property .

 בהרבה מקרים ננצל את ה- Properties לצורך בדיקות תקינות - סינון המידע הזורם אל מחוץ לגבולות המחלקה מתוך בלוק get , ונבדוק את המידע המגיע לבלוק set לפני ביצוע ההשמה בפועל.

 חשיבותם של ה- Properties היא בנוחות השימוש במחלקה, במתכנת העובד עם מחלקה המוגדרים בה Properties  מקננת התחושה שהוא עובד ישירות עם ה- Data Members , הקוד הרבה יותר קריא וקל לתחזוקה.

פירוט והרחבה:

ב-C# ניתן להגדיר משתנים ופונקציות כ-private או public. השימוש שהדגמנו בו הגדרנו משתנים כ-private, ולאחר מכן הגדרנו פונקציות get ו-set על מנת לגשת אליהן, הוא נפוץ ביותר, ולכן הוחלט להוסיף ל-C# מבנה דקדוקי מיוחד, המאפשר להאיץ תהליך זה.

מבנה זה הוא מנגנון ה-properties, שמאפשר לנו צורת עבודה המזכירה עבור משתמש הקצה עבודה מול שדות ישירות, כאשר בפועל יקראו פונקציות get ו-set עם כל קריאה של המשתמש.

נשפר שוב את המחלקה CRectangle, ונשתמש בתחביר החדש. נציג גם כיצד הפונקציה Main() משתנה.

using System;  
  
public class CRectangle  
{  
      private int m\_height, m\_width;  
  
      public int height  
      {  
            get  
            {  
                  return m\_height;  
            }  
            set  
            {  
                  if (value > 0) m\_height = value;  
            }  
      }

      public int width  
      {  
            get  
            {  
                  return m\_width;  
            }  
            set  
            {  
                  if (value > 0) m\_width = value;  
            }  
      }  
  
      public CRectangle()  
      {  
            m\_height = m\_width = 10;  
      }     
}  
  
public class CClassExample2  
{  
      public static int Main(string[] args)  
      {  
            CRectangle rec1, rec2;  
            rec1 = new CRectangle();  
            Console.WriteLine("Rectangle size: {0} x {1}",  
                  rec1.height, rec1.width);  
            rec1.height = 10;  
            rec1.width = 20;  
            rec2 = rec1;  
            Console.WriteLine("Rectangle size: {0} x {1}",  
                  rec2.height, rec2.width);  
            rec2.height = 15;  
            Console.WriteLine("Rectangle size: {0} x {1}",  
                  rec1.height, rec1.width);  
            return 0;  
      }  
}

במחלקה CRectangle הגדרנו משתנה עם הרשאת private בשם m\_height, וכן property בשם height. בהגדרת ה-property הגדרנו שני בלוקים, המשמשים כשתי פונקציות - get ו-set.

בפונקציה get החזרנו את המשתנה  m\_height, ובפונקציה set שיננו משתנה זה, אחרי שבדקנו את תקינות הקלט.

היתרון של השימוש ב-property הוא שכלפי מי שמשתמש באובייקט, ה-property מתנהג כמו משתנה, ועבור כותב המחלקה ה-property מתנהג כמו שתי פונקציה, כך שניתן לבצע בדיקות לפני שינוי הפרמטרים.

לאחר הקומפילציה של הקוד התכונות ממומשות כפי שממושו ב-C++, מתודת setPropertyName ומתודת GetPropertyName.

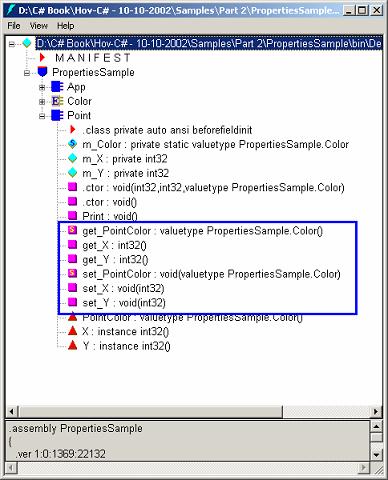
בנוסף חשוב לציין שמתכנתים רבים חוששים מהבדל מסויים בין C++ ו-C# והוא חוסר היכולת לקרוא למתודה בצורה inline (בצורה כזאת שבקוד יש קריאה לפוקנציה, אבל בפועל בקומפילציה המימוש מועתק לכל המופעים של הקריאה לצורך חסכון בתנועה במחסנית), ולכן חוששים מהשימוש במתודות ותכונות עבור שדות פשוטים, חשוב לציין עבור מתכנתים חסכניים אלו שבשפת C# כל תכונה שהביצוע שלה פשוט תהפוך לכזו לאחר תרגום קוד ה-IL לשפת מכונה (על ידי מנגנון JIT).

השימוש ב- Property :

Obj.PropertyName = SomeValue;

ללא Property :

Obj.SetPropertyName(SomeValue);



Properties נותנים אמנם תחושה של גישה ישירה לחלקים הפרטיים של המחלקה, אולם זו תחושה מוטעית, הם אינם Data Members אין להם כתובות זיכרון לאחסון המידע. למעשה הם מתקמפלים לפונקציות.

 כאשר הקומפיילר מהדר Property ל- MSIL, הוא ממיר את ה- Property לשתי המתודות כפי שהוסבר בתחילת הפרק.

ניתן לראות זאת באמצעות תוכנית השירות ildasm.exe :

* לחצן התחל או מקש win
* הקלידי CMD
* בחרי את

developer command prompt for vs2022

* בחלון ה- CLI שנפתח הקלידי ILDASM
* יפתח חלון זה:

 מאפיין אוטומטי – auto property

כדי שהקוד לא יראה מדי ארוך, כאשר אין בדיקות תקינות או שינויים קוסמטיים לערך המאפיין, C# מאפשרת לכתוב מאפיינים בצורה מקוצרת, כך:

Public int Id {get; set}

המשמעות היא שהמהדר ==הקומפיילר יוצר את השדה הפרטי מאחורי הקלעים בשם לא ידוע לנו ואין לנו אפשרות להשתמש בו אלא רק במאפיין. מסיבה זו לא ניתן להשמיט את אחד מהמנגישים (get or set ) אך ניתן לציין עבורו private כדי שיהיה נגיש רק בתוך המחלקה.

קיצורים:

prop עבור מאפיין אוטומטי

Propf עבור מאפיין מלא (full property)

ניתן להפוך שדה פרטי למאפיין על ידי לחיצה ימנית על השדה ובחירה באפשרות Quick actions and refractoring ואז לבחור ב- encapsulate field

מאפיין מחושב:

לא חובה להצמיד Data Member ל- Property , Property יכול לייצג משתנה חישובי אשר לא יופיע בפועל כ- Data Member במחלקה, בכל פעם שנפעיל את אותו ה- Property יתבצע החישוב מחדש ולכן תמיד נקבל ערך עדכני.

שדה חישובי מוגדר כשדה אשר ערכו הנו תוצאה של חישוב מתמטי המתבצע על ערכיהם של שדות אחרים הקיימים בתוכנית, מומלץ להימנע ככל הניתן מהכללת שדות חישוביים בתוכנית. הכללתם מסוכנת, משום שיש לעדכנם בכל פעם שאחד השדות המשמש כפרמטר בנוסחה משתנה, אם החישוב אינו מתבצע , ערכו של השדה החישובי אינו נכון דבר המוביל לשגיאות לוגיות.

ולכן , Data Member לעולם לא יהיה שדה חישובי (לפחות מומלץ להימנע מכך).

הפיתרון ש- Property מספק הוא הגדרת שדה חישובי כ- Property Readonly,

ה- Property יחשב מחדש את ערכו של השדה החישובי בכל פעם שהוא יופעל, בדרך זו נימנע מאותן השגיאות הלוגיות.

לדוגמא: כאשר יש שדה המכיל תאריך לידה וניצור מאפיין המחזיר את הגיל או את שנת הלידה.

יש שדות של שם פרטי ושם משפחה ונחזיר שם מלא.

תרגיל - (\*\*הקפידי על שמות תקינים\*\*)

כתבי מחלקת Food שתכיל:

משתנה פרטי id של המזון . קבלי את ערכו בבנאי.

משתנה ציבורי המכיל את השם שלו.

מאפיין מלא עבור מספר הקלוריות ל-100 גר'

מאפיין אוטומטי המציין האם מכיל שומן בכמות גבוהה. לא יהיה ניתן לשנות את הערך מחוץ למחלקה.   
כיצד עשית זאת? ......................

מאפיין אוטומטי על ידי הקיצור prop המציין האם המוצר מכיל נתרן בכמות גבוהה

מאפיין מלא לקריאה בלבד המציין האם המוצר מכיל סוכר בכמות גבוהה. כיצד עשית זאת? ......................

מאפיין מלא שנכתב באופן אוטומטי ע"י propf המציין את כמות השומנים במוצר.

מאפיין אוטומטי ע"י prop המציין את כמות השומנים הרוויים במוצר.

מאפיין מקוצר (אוטומטי) המכיל את זמן חיי המדף של המוצר (מסוג timespan )

מאפיין מחושב:

הוסיפי מאפיין (מחושב) של שם מלא המחזיר את שם המוצר והקוד שלו.

הוסיפי לערך המוחזר מ- get של המאפיין שם המוצר \*\* שתי כוכביות אם המוצר מכיל שומן בכמות גבוהה.

בתכנית הראשית (main ) צרי מוצר והשתמשי בכל המאפיינים שכתבת.

קמפלי.

פתחי את הקוד המקומפל על ידי ildasm ובחני את הפונקציות שנוצרו.

מה יקרה בהרצת הקוד הבא:

String name

Public string Name

{

get {return Name;}

}

האם למאפיין מחושב יש set accessor ? אפשר? ..... חובה? ...... אסור?...... חסר משמעות? ..........

באיזה מקרה תווצר שגיאת זמן ריצה של גלישת מחסנית ?

..................................................................................................................................

ציני נכון או לא נכון? וכן אם תגרם שגיאה- ציני: שגיאת קומפילציה או שגיאת זמן-ריצה ?

שם המאפיין חייב להיות זהה לשם השדה הפרטי? ............................................................

שם המאפיין חייב להתחיל באות גדולה ............................................................

במאפיין מקוצר חובה גם set וגם get............................................................

במאפיין מלא חובה גם set וגם get............................................................

אם אין בדיקות תקינות עדיף תיצור משתנה ציבורי ולא מאפיין............................................................