מבוא למדעי המחשב – כתה יא-1

תאריך: 12.2.2026

תכנות מונחה עצמים – עקרון הכמיסה (או כימוס) Encapsulation + תירגול #1

מבוא:

ננסה להבין דרך מספר דוגמאות פשוטות מה זה "קוד שנשבר", למה זאת בעיה חשובה, ואיך עקרון הכימוס עוזר לנו למנוע אותה.

תוך כדי כך נבין את הסיבה שבגללה יש `private / public` שבהם השתמשנו עד היום, אבל בלי להבין ממש לעומק.

מטרות

1. להבין *למה* לא ניגשים לשדות במחלקה באופן ישיר (public/private) – אפילו אם אנחנו יודעים את השמות של המשתנים.
2. לראות איך שינוי במימוש פנימי **שובר** קוד חיצוני כשלא מקפידים על זה.
3. ללמוד לבנות **ממשק ציבורי יציב** למחלקה, כך שקוד חיצוני לא יישבר.

נעבור דרך מספר תחנות – בכל אחת מהן יש קובץ עם קוד (שמות הקבצים הם במתכונת של `Pet_example_1.cs`). לעתים נראה את הבעיה ולעתים נראה את הפתרון לבעיה בתחנה הקודמת.

הערה:

בכל פעם שנרצה לעבוד עם קובץ אחר ב `Vistual Studio` נשתמש בתפריט של `solution explorer` ויש שם אפשרות לעשות `include in project or exclude from project.` חשוב שתתנסו בזה (אם זה לא מובן תשאלו את צבי, אחרת סתם עבודה קשה למשל, לפתוח בכל פעם פרוייקט חדש – זה לא עוזר לחומר הלימודי).

הקבצים נמצאים בתיקייה הזאת.

<u>מושגים שמוכרחים להבין</u>

(הדוגמאות בהמשך יעזרו לנו להבין אותם)

- **מימוש פנימי (Internal Implementation):** ה"סוד המקצועי" של המחלקה. זהו האופן שבו בחרנו לייצג את הנתונים בתוך המחלקה (השדות) ואיך הפעולות עובדות "מתחת למכסה המנוע". "העולם שבחוץ לא אמור לראות או להכיר את החלק הזה.
- **ממשק ציבורי" (Public Interface / API):** תפריט השירותים" שהמחלקה מציעה לעולם. אלו הן כל הפעולות והבנאים שהגדרנו כ-public. זהו החלק היחיד שדרכו מותר למחלקות אחרות לתקשר עם האובייקט שלנו.
- **כְּמִיסָה (Encapsulation):** אריזת הנתונים (השדות) והפעולות ליחידה אחת סגורה, תוך **הסתרת המימוש הפנימי**. המטרה היא להגן על המידע כך שגישה אליו תתבצע אך ורק דרך ה"שערים" (הפעולות הציבוריות) שהגדרנו בממשק.
- **תלות (Coupling):** מצב שבו מחלקה אחת "יודעת יותר מדי" על המבנה הפנימי של מחלקה אחרת. כשיש תלות גבוהה, כל שינוי קטן במימוש הפנימי של מחלקה א' יגרום ל"פיצוץ" (שגיאת קומפילציה) בכל המחלקות שתלויות בה [זה מצב לא רצוי].

תחנה 1 – "דוגמא רעה" (Pet_example_1.cs)

מחלקה Dog, מחלקה Vet (ווטרינר באנגלית)

```
class Dog
{
    public string name;  // BAD: public field
    public int age;      // BAD: public field (years)

    public Dog(string name, int age)
    {
        this.name = name;
        this.age = age;
    }

    public string ToInfo()
    {
        return "Dog: " + name + ", age=" + age;
    }
}

class Vet
{
    private Dog dog;

    public Vet(Dog dog)
    {
        this.dog = dog;
    }

    public void PrintDog()
    {
        // Vet touches Dog's internal data directly:
        Console.WriteLine("[VET] " + dog.name + ", " + dog.age + " years old");
    }

    public void HaveBirthday()
    {
        Console.WriteLine("** Happy Birthday to: " + dog.name + " !! **");
        dog.age++; // direct change
                   // change age by a year
    }
}

public partial class Program
{
    static void Main()
    {
        Dog d = new Dog("Luna", 3);
        Vet v = new Vet(d);

        v.PrintDog();
        Console.WriteLine(d.ToInfo());

        v.HaveBirthday();
        v.PrintDog();

        Console.WriteLine(d.ToInfo());
    }
}
```

שאלות בקשר לקוד הזה:

1. למה זה "רע" שמחלקת Vet נוגעת ישירות ב־dog.age וב־dog.name? כתוב 2 סיבות.
2. מי אמור להיות "האחראי" על כך שהגיל לא יהיה שלילי? האם זאת המחלקה Dog או המחלקה Vet? למה ?
3. האם Vet יודע יותר מדי על Dog? מה בדיוק?

תחנה 2 – שינוי מימוש ששובר קוד (Pet_example_2.cs)

המטרה: להרגיש בעצמכם (דרך הידיים) מה קורה כשמשנים ייצוג פנימי.

משימה:

עברו לקובץ Pet_example_2.cs. (בצעו **Exclude** לקובץ הקודם ולאחר מכן **Include** לקובץ זה).

נסו לקמפל ולהריץ.
תגלו שהקוד אינו מתקמפל.

1. קראו בעיון את הודעות השגיאה שהקומפיילר מציג.
2. זהו מהן הבעיות שגורמות לקוד להישבר.
3. רשמו במחברת:
   - אילו שורות בקובץ Vet גורמות לשגיאה?
   - מה הקשר בין השגיאה לבין השינוי שביצענו במחלקה Dog?
4. תקנו את המחלקה Vet כך שהקוד יתקמפל וירוץ שוב. (בשלב זה מותר "לתקן מהר", גם אם הפתרון אינו אידיאלי).

שאלה לחשיבה:

אם בפרויקט היו 100 מחלקות שמשתמשות במחלקה Dog – כמה מקומות היינו צריכים לתקן עכשיו?

<u>תחנה 3 — תיקון נכון: כימוס</u>

המטרה: לשנות את המחלקה `Dog` כך ששינוי במימוש הפנימי שלה לא ישבור מחלקות אחרות.

משימה:

עברו לקובץ `Pet_example_3.cs`. (בצעו **Exclude** לקובץ הקודם ולאחר מכן **Include** לקובץ זה).

בקובץ תמצאו הערות הנחיה (`TODO`) המסבירות מה יש לשנות.
תקנו את הקוד לפי ההנחיות.

עבדו לפי הסדר הבא:

1. הפכו את השדות של המחלקה `Dog` ל־`private`.
2. הוסיפו פעולות ציבוריות מתאימות:
   - פעולה מאחזרת המחזירה את שם הכלב.
   - פעולה מאחזרת המחזירה את הגיל בשנים (גם אם הגיל נשמר פנימית בחודשים).
   - פעולה מעדכנת שמוסיפה שנה לגיל.
3. עדכנו את המחלקה `Vet` כך שלא תיגש לשדות של `Dog` ישירות, אלא תשתמש רק בפעולות הציבוריות שהוספתם.

לאחר ביצוע כל השלבים:

4. קמפלו והריצו את התוכנית.
5. ודאו שהקוד עובד כראוי.

בדיקה עצמית:

- האם ניתן כעת לשנות שוב את הייצוג הפנימי של `Dog` (למשל לשנות את פורמט השם המוחזר) מבלי לשנות את המחלקה `Vet`?

סיכום קצר במחברת (3–4 שורות):

נסחו במילים שלכם מהו כימוס, ומה הרווחנו ממנו בדוגמה זו.

<u>תחנה 4 — הוספת מחלקה Cat והרגשת הכאב</u>

המטרה: להרגיש מה קורה כשיש שתי מחלקות כמעט זהות שצריך לתחזק.

<u>משימה - חלק א – השלמת Cat</u>

עברו לקובץ Pet_example_4.cs, (בצעו **Exclude** לקובץ הקודם ולאחר מכן **Include** לקובץ זה).

בקובץ תמצאו:

- את המחלקה Dog (כפי שהיא בסוף תחנה 3).
- שלד ריק למחלקה Cat
- פעולה Main_Dog_And_Cat()

השלימו את המחלקה Cat כך שתהיה דומה מאוד ל- Dog :

1. הגדירו שדות פרטיים מתאימים (שם, סוג, גיל בחודשים).
2. כתבו **פעולה בונה** תואמת: Cat(string name, int ageInYears)
   - השם נשמר פנימית
   - סוג ברירת מחדל יהיה "Cat"
   - הגיל יישמר פנימית בחודשים
3. הוסיפו פעולות:
   - GetName()
   - GetAge() בשנים
   - HaveBirthday()
   - ToInfo()

לאחר מכן הריצו את Main_Dog_And_Cat() וודאו שמודפס מידע גם לכלב וגם לחתול.

<u>משימה (חלק ב – שינוי קטן ומעצבן):</u>

בצעו שינוי קטן *בשתי המחלקות*:

- שנו את הפורמט של הפעולה המאחזרת GetName() כך שיחזיר:
  - "Species - Name"
    לדוגמה:: Dog - Luna או Cat - Mizi

לאחר השינוי:

1. בדקו שהקוד עדיין מתקמפל ורץ.

2. כתבו במחברת:
   - כמה מקומות הייתם צריכים לשנות?
   - מה יקרה אם יהיו לנו 6 סוגי חיות?

לא פותרים את הבעיה היום. רק מרגישים את הכאב.