מבוא למדעי המחשב – כתה יא-1

תאריך: 14.12.2025

דף עבודה והסברים קצרים – מספר #5

מבוא כללי לתכנות גנרי

רשימות מקושרות גנריות (Linked Lists using Node<T>)

(עדכון - גרסה #2 27.12.25)

מבוא

התרגיל הזה מבוסס על כך שאנו יודעים כבר לעבוד עם רשימות מקושרות מהסוג שראינו בתרגילים ודפי עבודה הקודמים.

בתרגיל הזה, "נתקרב מאוד" – כלומר, נבין את הצורך בשימוש במחלקה-חוליה שהיא מוכנה מראש, כלומר היא חלק מהשפה.

מוכנה מראש: מישהו כבר הגדיר איך היא נראית, ומימש את הפעולות (פונקציות) שהיא יודעת לעשות.

ואנחנו לא צריכים לכתוב את המחלקה הזאת!

בחלק א' של התירגול הזה – שמופיע כאן – נבין את הצורך, והוא הכנה לשיעור שבו ממש נלמד את הנושא.

בשיעור שלאחר התירגול, נלמד איך לכתוב תכנות גנרי. כלומר איך להשתמש במחלקה-חוליה שהשפה מאפשרת להגדיר, ומסתמכים עליה בבגרות. המחלקה הזאת נקראת Node<T> עבור רשימות מקושרות.

מה שאנחנו צריכים לעשות: זה לדעת להשתמש בה! לדוגמא, לעשות את כל הפעולות שעשינו בתרגילים הקודמים – אבל מבלי שנכתוב בעצמנו את החוליה (כמו למשל IntNode), וגם, חלק מהפעולות שעשינו - נעשה אותן בצורה נוחה יותר (נכתוב פחות קוד בעצמנו) כי החוליה הזאת כבר מספקת פעולות שונות.

בחלק ב' של התירגול – בשבוע הבא, כבר נכתוב בעצמנו את המחלקה הזאת, ונכתוב קוד גנרי שמשתמש בה.

עד כאן המבוא לתירגול ולשיעור של היום.

ועכשיו, לתירגול עצמו.

הכנות

פתחו פרוייקט חדש ב Visual Studio. הורידו אליו מהקישור באתר (ה GDRIVE) את הקבצים הבאים:

```
Node_of_Int.cs
Node_of_String.cs
Program_int.cs
Program_main_student.cs
Program_string.cs
```

קמפלו והריצו... זה אמור לעבוד אם כי תקבלו הודעות על פונקציות לא ממומשות.

תזכורת: דברים שלמדנו וראינו בתירגול מספר #3:

- שימוש בחוליה מקשרת
- הפרדה בין החוליה המקשרת למחלקה-אובייקט שמכיל את הנתונים
- עקרון הפרדת האחראיות – מחלקה אחת היא החוליה שאחראית על הקישור (IntNode) ומחלקה אחרת אחראית על המידע (IntData).

משימה I – להבין את המוטיבציה - למה צריך תכנות גנרי בהקשר של מערכים

תארו לעצמכם שיש לנו שלושה מערכים מטיפוסים שונים –

```
int[] iArr = {10, 11, 12, 13};
double[] dArr = {2.5, 3.5, 4.5};
string[] sArr = {"avi", "beni", "gila", "dana"};
```

לצורך הפשטות נגדיר את המערכים האלו בצורה סטטית, כלומר כחלק מהקוד, בלי לקלוט שום דבר. קובץ קובץ שלד לתרגיל הזה נמצא בקובץ Program_main_student.cs והוא מכיל כבר את הקוד הזה.

<u>תרגיל 1</u>

עכשיו אנחנו רוצים לכתוב לפלט את המערכים השונים. האיברים של כל מערך נכתבים בצורה זהה, כולם בשורה אחרת כאשר הם מופרדים בפסיק + רווח.

כל הדפסה תבוצע על ידי פונקציה שמקבלת את המערך.

ממשו את הפונקציות האלו (כלומר: כתבו את הקוד) והריצו את התכנית. (אכן, זאת תכנית פשוטה שאתם יכולים לכתוב בעיניים עצומות כמעט).

<u>הנחייה:</u>

אל תשכחו להיות "עצלנים": מכיוון שהפונקציות דומות מומלץ לכתוב פונקציה אחת, להריץ ולוודא שהיא עובדת היטב, ואז להעתיק את הקוד שלה, ולכתוב את האחרות, תוך ביצוע השינויים הדרושים.

שמות מומלצים לפונקציות הם PrintIntArr, printDoubleArr, PrintStringArr.

פלט לדוגמא עבור מערך המחרוזות:

```
avi, beni, gila, dana
```

שימו לב שאין פסיק אחרי האיבר האחרון אלא ירידת שורה.

<u>רפלקציה על תרגיל 1</u>

רפלקציה זה כשאנחנו עוצרים לרגע תוך כדי העשייה של משהו, או מיד לאחר שעשינו, ושואלים את עצמנו שאלות כמו: איך זה היה? האם זה קל? האם הגישה הזאת טובה? מה יקרה אם יהיו לנו גם מערכים של טיפוסים נוספים (למשל בוליאני)? מה יקרה אם יהיה "*שינוי קטן בדרישות*" (כמו למשל בתרגיל הבא).

כתבו 2-3 משפטים במחברת (לא חייבים לכתוב את כל מה שעולה בדעתכם, רק חלק קטן).

<u>תרגיל 1-א</u>

בתרגיל הקודם יצרנו שלוש פונקציות שונות, PrintIntArr, printDoubleArr, PrintStringArr כאשר כל אחת מהפונקציה מקבלת טיפוס שונה – מערך של int או double או string.

כעת, נרצה לשנות את שמות הפונקציות שיהיו בדיוק אותו שם לדוגמא :

```
public static void PrintArr(int[] arr)

public static void PrintArr(double [] arr)

public static void PrintArr(string [] arr)
```

לפני שאתם מנסים את השינוי הזה עצרו וחישבו: האם זה אפשרי? האם זה יעבור קומפילציה?

נסו את השינוי, ואז עצרו והסבירו – מה בעצם קורה כאן?

<u>תרגיל 2</u>

כעת אנו רוצים שהפלט יהיה מעט שונה – באופן הבא: כל מערך יודפס בתוך סוגריים מסולסלות.

המשימה שלכם: לשנות את הפונקציות שייצרו את הפלט לפי הדרישה החדשה.

פלט לדוגמא עבור מערך המחרוזות והמערך של השלמים:

```
{avi, beni, gila, dana}
{10, 11, 12, 13}
```

כאשר סיימתם לכתוב את הקוד (לתקן אותו) בצעו שוב רפלקציה:

<u>רפלקציה על תרגיל 2</u>

- האם זה קל?
- האם הגישה הזאת טובה?
- האם היה בתרגיל משהו מעצבן?
- מה יקרה אם ידרשו מאיתנו לעשות שינוי נוסף, למשל במקום סוגריים מסולסלים, להדפיס את המערכים בין סוגריים מרובעים? או אולי, לכתוב אינדקס לפני כל איבר לדוגמא:

```
{ [0] avi [2] beni [3] gila [4] dana }
{ [0] 10 [1] 11 [2] 12 [3] 13 }
```

<u>שאלות כללית</u>

- נניח שהייתה לנו דרך לכתוב רק פונקציה אחת במקום שלוש, שתדע לטפל בשלושת המערכים השונים, האם זה היה מקל עלינו?
- מה בעצם הסיבה שאנחנו לא יכולים לעשות את זה? (הרי מבחינת הלוגיקה, כלומר הקוד ואיך שהוא מתנהג) זה אותו הדבר בשלושת המקרים?
- <u>הערה</u>: בקובץ הדוגמא שמסופק לכם `Program_main_student.cs` מופיעה כבר הפונקציה הבאה: אתם מוזמנים לעיין בא מתוך סקרנות – זה יוסבר בשיעור. אתם יכולים לנסות להוסיף לקוד (לפונקציה `ArraysDemo()`) קריאה לפונקציה הזאת עם כל אחד מהמערכים שהגדרנו.

בינתיים - נשאיר את כל השאלות (הרפלקציות) פתוחות (כלומר ללא מענה), ונעבור לעשות טיפול דומה ברשימות מקושרות.

<u>משימה II – להבין את המוטיבציה - למה צריך תכנות גנרי בהקשר של רשימות</u>

<u>תרגיל #3 – משימות בסיסיות עם רשימה של מספרים שלמים</u>

כעת נרצה לבנות רשימה של מספרים שלמים חיוביים. נקלוט אותם מהקלט. כאשר נקלוט מספר שלילי, זה מציין את סיום הקלט.

לצורך הפשטות, נכניס את האיברים החדשים לראש הרשימה.

נבנה מחלקה של חוליה של `int` שנקרא לה `Node_of_Int`. והפעם, נקפיד על עקרון ההכמסה. כלומר יהיו לנו משתני מחלקה שהם `private` ופונקציות מחלקה שהן `public`.

(הערה: אם אתם זוכרים, בתרגילים עם רשימות מקושרות שהיו לנו עד כה, הכל היה `public`. זה לא בגלל שככה נכון לעבוד, אלא במטרה להקל עלינו כשלמדנו נושא מורכב וחדש. אבל – זאת הייתה "הנחה" זמנית, והמבצע של `black Friday` הסתיים 😊). חוזרים לתכנת בצורה מוקפדת.

בצעו את המשימות הבאות:

א. **הגדירו מחלקה בשם Node_of_Int**. להלן הגדרה חלקית של המחלקה, השלימו אותה כולל הפונקציות Get/Set הדרושות. (כמובן, Visual Studio יציע לכם ליצור אותה בקובץ נפרד שזה גם השם שלו – עשו כך). (הערה: בהנחה שעבדתם על פי ההוראות בתחילת התרגיל, הקובץ הזה עם חלק מהקוד כבר נמצא שם).

```
public class Node_of_Int
{
    private int value;
    private Node_of_Int next;

    // בנאי לחוליה אחרונה (ללא next)
    public Node_of_Int(int value)
    {
        this.value = value;
        next = null;
    }

    //  הוצא מהערה והשלם את הבנאי הבא
    // זהו בנאי לחוליה עם ארגומנט שהוא ההפניה לאיבר הבא
    //public Node_of_Int(int value, Node_of_Int next)
    //{
    //}

    public int GetValue()
    {
        return value;
    }

    public Node_of_Int GetNext()
    {
        return next;
    }

    // code you have to complete
}
```

ב. קראו בעיון את המחלקה. שימו לב למשתנים הפרטיים, ושימו לב לפונקציות הציבוריות. בפרט – מה תפקידה של הפונקציה HasNext ? האם אפשר להסתדר בלעדיה? אם אפשר אז למה הגדירו אותה?

ג. צרו קובץ שנקרא Program_int.cs. (בעצם, יש לכם קובץ בסיס בקישור על ה GDRIVE). הוא יהיה הרחבה של המחלקה Program ולכן הוא יכיל בתור בסיס את הקוד הבא:

```
public partial class Program
{
    public static void PrintList_of_Int(Node_of_Int chain)
    {
        // you need to implement this
    }


    public static Node_of_Int BuildAndPrintList_of_Int()
    {
        Node_of_Int list = BuildList_of_Int();
        PrintList_of_Int(list);
        return list;
    }
```

ד. עליכם לכתוב פונקציה שקולטת מספרים שלמים חיוביים ומכניסה אותם לרשימה. סוף הקלט הוא מספר שלילי. לאחר מכן הרשימה מודפסת. להלן הקריאה מה MAIN וגם הפונקציה עצמה:

```
public static Node_of_Int BuildAndPrintList_of_Int()
{
    Node_of_Int list = BuildList_of_Int();
    PrintList_of_Int(list);
    return list;
}

// and the call from main:
    Node_of_Int myList = BuildAndPrintList_of_Int();
```

ה. אני משער שבקוד של הפונקציה BuildList_of_Int() היו לכם 3 שורות כאלו:

```
// האם אנו יכולים להחליף את שתי הפקודות הבאות בפקודה אחת
Node_of_Int newNode = new Node_of_Int(num);
newNode.SetNext(head);
head = newNode;
```

כלומר, את השורה head = newNode; נשאיר כפי שהיא, אבל את שתי השורות שלפני נאחד לשורה אחת.

<u>הנחיה:</u>

מה שדרוש בכדי לעשות את זה – שיהיה לנו בנאי שהקריאה אליו מציבה למשתנה next את הערך של התא הבא. הוסיפו את הבנאי הזה לקוד שלכם, שנו את השורות הנ"ל בפונקציה BuildList_of_Int והריצו מחדש בכדי לוודא שהכל עובד כראוי.

```
// בנאי לחוליה עם מצביע ל-next
public Node_of_Int(int value, Node_of_Int next)
{
    this.value = value;
    this.next = next;
}
```

<u>רפלקציה על תרגיל #3</u>

האם למדנו כאן משהו חדש?

האם יש בעייה בכך שלמחלקה יש שני בנאים?

איך זה דומה למה שראינו קודם – פונקציות שונות עם אותו שם – PrintArr ?

איזה מחשבות עולות לכם?

<u>נתבונן בשם של המחלקה-חוליה Node_of_Int</u>

האם השם של המחלקה של החוליה Node_of_Int נראה לכם קצת מסורבל מדי? זה גם לא בדיוק הקונבנציה (מה שמקובל) בשפת C# להשתמש בקו-תחתון – אבל זה חוקי. ועוד מעט נבין את הסיבה למה אני (צבי) בחרתי להשתמש בצורה כזאת לשם של הפונקציה.

<u>תרגיל 3-א רשות (לעשות בבית):</u>

בצעו את אותה משימה כאשר מוסיפים כל איבר חדש בסוף הרשימה. לשם כך תהיה לכם פונקציה כזאת:

```
{
    Node_of_Int list = BuildListAtTail_of_Int();
    PrintList_of_Int(list);
    return list;
```

```
        }

    public static Node_of_Int BuildListAtTail_of_Int()
    {
        Node_of_Int head = null;
        Node_of_Int tail = null;

        int num = int.Parse(Console.ReadLine());

        while (num >= 0)
        {

       // השלימו את הקוד בעצמכם
```

תרגיל #4

התרגיל הזה דומה מאוד לתרגיל #3.

גם כאן נבנה רשימה אבל הפעם זאת תהיה רשימה של מחרוזות.

א. לשם כך נגדיר את המחלקה הבאה `public class Node_of_String`. המחלקה הזאת היא חוליה מקשרת בין מחרוזות. כך נראות השורות הראשונות שלה:

```
public class Node_of_String
{
    private string value;
    private Node_of_String next;
```

ב. עליכם ליצור בתכנית ה MAIN את הקטע הבא (או להוציא מהערה אם הקטע הזה כבר קיים)

```
Console.WriteLine("Enter strings (empty string to stop):");
Node_of_String list = BuildList_of_String();

// אפשר להדפיס בעזרת פונקציה קיימת
PrintList_of_String(list);
```

<u>הנחיות:</u>

1. קובץ אחד שעליכם ליצור בשביל החוליה הוא `Node_of_String.cs`
2. קובץ שני שאתם צריכים, השלד שלו נמצא בקישור הנתון על ה GDRIVE (כרגיל). זהו הקובץ `program_string.cs`.
3. בנוסף, כפי ששמתם לב נתונים לכם הקבצים `program_int.cs` שבו אתם כותבים את הקוד של הפונקציות שעוסקות ברשימה של `int` והקובץ `program_main_student.cs` שמכיל את ה MAIN של התכנית, ובו עליכם לממש את תרגילים #1 + #2 (הדפסות ע"י מערכים).
4. הקבצים שעליכם ליצור בעצמכם הם `Node_of_Int.cs` – עבור חוליה של מספרים שלמים. זה מה שנחוץ לכם לתרגיל #3, ובתרגיל #4 עליכם ליצור קובץ-מחלקה `Node_of_String.cs`.
5. ביצירת הקובץ `Node_of_String.cs` אני ממליץ לכם "להיות עצלנים" – להעתיק את תוכן הקובץ `Node_of_Int.cs` ולעשות בו שינויים הכרחיים מינימליים. שימו לב, מה בעצם אתם משנים. האם אתם צריכים לעשות שינויים גדולים או קטנים? האם אפשר לעשות את השינויים הנדרשים ע"י פעולת "מצא-החלף" – `find-replace` (אגב יש לה קיצור: `Cntl+H`).

<u>שאלה מסכמת: אז מה זה אומר "תכנות גנרי" ולמה צריך את זה?</u>

ראינו מחלקה שהיא חוליה מקשרת לרשימה של מספרים שלמים. וראינו מחלקה שהיא חוליה מקשרת לרשימה של מחרוזות. הן פועלות וכתובות בדיוק אותו הדבר, בהבדל אחד: הראשונה עובדת על טיפוס `int` והשנייה עובדת על טיפוס `string`.

בעצם, יכולנו לקחת את המחלקה האחת להעתיק אותה ורק להחליף את הטיפוס `int` בטיפוס `string` . פשוט להחליף מילה במילה. זה כל ההבדל.

יתר על כן – נניח שאנחנו רוצים לבצע מעבר על רשימה של `int` ולספור כמה איברים היא מכילה או לבצע מעבר על רשימה של מחרוזות, ולספור כמה איברים היא מכילה – זאת תהיה בדיוק אותה פעולה, ואנחנו שואפים לכך שפונקציה אחת תוכל לבצע את המשימה בשני המקרים.

נרחיב על הנושא בשיעורים בשיעור בכיתה ובתירגולים הבאים.