Exercise #1 – Intro to C#

Advanced programming 2 (2019)

במטלה זו אנו נתרגל את כל הנושאים שלמדנו בתרגולים + הרצאות, כגון:

Delegate, Event, List, Dictionary and Lambda Expressions.

אנא שימו לב כי אתם משתמשים בנושאים אלו בתרגיל הנוכחי ובמהלך הסמסטר.

Function Container, Single Mission and Combined Mission

במטלה זו נרצה להשתמש ב- Design patterns אשר הכרנו בסמסטר הקודם:
Command & Builder

בתוכנית הבאה, אנחנו נרצה להגדיר מילון של פונקציות (כאשר המפתח של המשימה הינו מחרוזת והערך הינו מצביע לפונקציה) ולבנות אובייקטים של משימות (***משימה בסיסית*** – אשר מהווה הפעלה של פונקציה **יחידה**, ***ומשימה מרוכבת*** – אשר מהווה הרכבה של פונקציות)

את הקוד המלא תוכלו למצוא בקובץ Program.c בתוך הפונקציה main.

<u>את הפלט של הפעלת התוכנית תוכלו למצוא בקובץ Output.txt</u>

לפיכך במטלה זו עליכם לממש את המחלקות הבאות:

- **FunctionsContainer**. – מחלקה זו מכילה INDEXER אשר בהינתן שם של פונקציה (מחרוזת) מחזירה פעולה מתאימה המקבלת ומחזירה ערך יחיד **:** $f: \mathbb{R} \to \mathbb{R}$
- **SingleMission –** מחלקה זו מאחסנת פעולה אחת מבין הפונקציות אשר מאוכסנות במחלקה FunctionsContainer. על מנת להפעיל את המשימה יש לקרוא לפונקציה **Calculate** אשר קיימת במחלקה.
- **ComposedMission–** מחלקה זו מאחסנת שרשור של פונקציות מבין הפונקציות אשר מאוכסנות במחלקה FunctionsContainer. על מנת להפעיל את המשימה יש לקרוא לפונקציה **Calculate** אשר קיימת במחלקה.

Function Container

```
FunctionsContainer funcList = new FunctionsContainer();
funcList["Double"] = val => val * 2;
funcList["Triple"] = val => val * 3;
funcList["Square"] = val => val * val;
funcList["Sqrt"] = val => Math.Sqrt(val);
funcList["Plus2"] = val => val + 2;
```

בקטע קוד זה אנחנו יצרנו את המחלקה **FunctionsContainer** והוספנו אליה פונקציות ע"י שימוש בעקרון ה- **Lambda Expressions**. **(פונקציות בלבד מהסוג:** $f: \mathbb{R} \to \mathbb{R}$ **).**

כאשר נזין מחרוזת כ-index במחלקה זו אנחנו נקבל את התמודה המתאימה **(חשבו באיזה מבנה נתונים כדאי להשתמש, ומה הערכים אשר הוא יאחסן)**.

Mission Interface

כל משימה באשר היא מממשת את הממשק **IMission**. הממשק מכיל :

- מתודת ה- Calculate
- event אשר מופעל כאשר קוראים למתודה `Calculate`
- 2 שדות אשר מציינים את שם המשימה ואת הטיפוס שלה.

```
public interface IMission
{
        event EventHandler<double> OnCalculate;  // An Event of when a mission is activated

        String Name { get;}
        String Type { get; }

        double Calculate(double value);
}
```

נבחין כי המחלקות SingleMission ו- ComposedMission ממשות ממשק וזה ועל כן מחויבות להגדיר מימוש לכל דבר בקיים ב- Interface.

נתבונן על המשימות שיצרנו:

```
ComposedMission mission1 = new ComposedMission("mission1")
                .Add(funcList["Square"])
                .Add(funcList["Sqrt"]);
```

יצרנו משימה מסוג **ComposedMission** אשר מבצעת את החישוב הבא:

$$\sqrt{(x^2)}$$

```
ComposedMission mission2 = new ComposedMission("mission2")
                .Add(funcList["Triple"])
                .Add(funcList["Plus2"])
                .Add(funcList["Square"]);
```

יצרנו משימה מסוג **ComposedMission** אשר מבצעת את החישוב הבא:

$$\sqrt{((x * 3) + 2)}$$

```
SingleMission mission3 = new SingleMission(funcList["Double"], "mission3");
```

יצרנו משימה מסוג **SingleMission** אשר מבצעת את החישוב הבא:

$$2 * x$$

לפני שנתאר את המשימה הבאה, נבחין כי אשר נדפיס את הפונקציות הקיימות לנו נקבל כי:

```
All Available Functions:
Double
Triple
Square
Sqrt
Plus2
####################################
```

כעת, נתאר את המשימה הבאה:

```
ComposedMission mission4 = new ComposedMission("mission4")
                .Add(funcList["Triple"])
                .Add(funcList["Stam"])
                .Add(funcList["Plus2"]);
```

אולם, אם נבחין בקוד, נראה כי לא קיימת פעולה בשם Stam. לפיכך **מצופה מהמחלקה FunctionsContainer** פעולה חדשה ששמה Stam אשר **<u>לא משנה את ערכו של הערך שהתקבל</u>** אלא מחזירה אותו כפי שהוא.

$$Stam(x) = x$$

לפיכך המשימה אמורה לבצע את בחישוב הבא:

$$(x * 3) + 2$$

כעת, נבחין כי אשר נדפיס את הפונקציות הקיימות לנו נקבל כי:

```
All Available Functions:
Double
Triple
Square
Sqrt
Plus2
Stam
#####################################
```

<u>שים/י לב</u> כי אנחנו נוכל לשנות את stam בהמשך הקוד באופן הבא:

```
funcList["Stam"] = val => val + 100;
SingleMission mission5 = new SingleMission(funcList["Stam"], "mission5");
```

ונוכל ליצור משימה חדשה אשר מבצעת את החישוב הבא:

$$x + 100$$

(אולם, משימה 4 **<u>לא</u>** השתנתה עקב שינוי זה!)

Event Handlers – 3 ניתן להתחיל לפתור החל מתרגול מספר

אם נחזור לקוד המקורי, אנחנו נבחין כי יצרנו שני EventHandlers אשר נרשמים לאירוע OnCalculate עבור המשימות שניצור בעתיד.

נתבונן על כל EventHandler בנפרד:

```
EventHandler<double> LogHandler = (sender, val) =>
{
        IMission mission = sender as IMission;

        if (mission != null)
        {
                Console.WriteLine($"Mission of Type: {mission.Type} with the Name
                {mission.Name} returned {val}");
        }
};
```

יצרנו EventHandler אשר בהינתן הפעלת המתודה calculate כותב ב- Console את פרטי המשימה שהופעלה ואת הערך המוחזר שלה.

```
EventHandler<double> SqrtHandler = (sender, val) =>
{
        SingleMission sqrtMission = new SingleMission(funcList["Sqrt"], "SqrtMission");

        double newVal;
        do
        {
                newVal = sqrtMission.Calculate(val);      // getting the new Val
                Console.WriteLine($"sqrt({val}) = {newVal}");

                val = newVal;                            // Storing the new Val;
        } while (val > 2);
        Console.WriteLine("----------------------------------------");
};
```

יצרנו EventHandler אשר בהינתן הפעלת המתודה calculate הוא יוצר משימה של Sqrt אשר כל עוד הערך שהתקבל אינו קטן מ-2 הוא ימשיך לבצע את Sqrt.

מאחר ויש 2 Event Handlers שונים, נרצה לקשר אותם ל- Mission באמצעות ה- Event

```
var missionList = new List<IMission>() { mission1, mission2, mission3, mission4, mission5 };

foreach (var m in missionList)
{
        m.OnCalculate += LogHandler;
        m.OnCalculate += SqrtHandler;
}
```

Continue on Main

ונוסיף לרשימה מופעים נוספים של המשימות:

```
missionList.Add(mission2);
missionList.Add(mission1);
missionList.Add(mission3);
```

כעת, כאשר נפעיל את כל המשימות עם הפרמטר 100: `RunMissions(missionList, 100);`
נקבל:

```
Mission of Type: Composed with the Name mission1 returned 100
sqrt(100) = 10
sqrt(10) = 3.16227766016838
sqrt(3.16227766016838) = 1.77827941003892
----------------------------------------
mission1(100) = 100

Mission of Type: Composed with the Name mission2 returned 91204
sqrt(91204) = 302
sqrt(302) = 17.3781471969828
sqrt(17.3781471969828) = 4.16871049570281
sqrt(4.16871049570281) = 2.04174202476777
```

```
sqrt(2.04174202476777) = 1.42889538622244
----------------------------------------
mission2(100) = 91204

Mission of Type: Single with the Name mission3 returned 200
sqrt(200) = 14.142135623731
sqrt(14.142135623731) = 3.76060309308639
sqrt(3.76060309308639) = 1.93922744748686
----------------------------------------
mission3(100) = 200

Mission of Type: Composed with the Name mission4 returned 302
sqrt(302) = 17.3781471969828
sqrt(17.3781471969828) = 4.16871049570281
sqrt(4.16871049570281) = 2.04174202476777
sqrt(2.04174202476777) = 1.42889538622244
----------------------------------------
mission4(100) = 302

Mission of Type: Single with the Name mission5 returned 200
sqrt(200) = 14.142135623731
sqrt(14.142135623731) = 3.76060309308639
sqrt(3.76060309308639) = 1.93922744748686
----------------------------------------
mission5(100) = 200

Mission of Type: Composed with the Name mission2 returned 91204
sqrt(91204) = 302
sqrt(302) = 17.3781471969828
sqrt(17.3781471969828) = 4.16871049570281
sqrt(4.16871049570281) = 2.04174202476777
sqrt(2.04174202476777) = 1.42889538622244
----------------------------------------
mission2(100) = 91204

Mission of Type: Composed with the Name mission1 returned 100
sqrt(100) = 10
sqrt(10) = 3.16227766016838
sqrt(3.16227766016838) = 1.77827941003892
----------------------------------------
mission1(100) = 100

Mission of Type: Single with the Name mission3 returned 200
sqrt(200) = 14.142135623731
sqrt(14.142135623731) = 3.76060309308639
sqrt(3.76060309308639) = 1.93922744748686
----------------------------------------
mission3(100) = 200

Mission of Type: Single with the Name mission5 returned 200
sqrt(200) = 14.142135623731
sqrt(14.142135623731) = 3.76060309308639
sqrt(3.76060309308639) = 1.93922744748686
```

---
mission5(100) = 200

צורת עבודה:

את התרגיל מגישים **ביחידים** בלבד!
את התרגיל יש לעשות בעזרת SOURCE CONTROL ומומלץ להשתמש ב git בעזרת bitbucket.
גם אם אתם עובדים לבד, חובה לעבוד ולהשתמש ב- source control אשר צורת העבודה אתו נלמדה ותורגלה בקורס תכנות מתקדם 1.
את התרגיל כולו כותבים ב #C -אין להשתמש בספריות אשר מחייבות התקנה נוספת של דברים מעבר ל Net Framework. בפרט, לא ניתן להשתמש בספריות או קוד מוכן אשר עושה עבורכם דברים שאתם אמורים לממש.

לתרגיל ישנו פורום ייעודי **במודל**. את כל השאלות יש לשאול אך ורק דרך הפורום **במודל**.
חובה להתעדכן בפורום. כל הנאמר בו מחייב את כולם.
עליכם לכתוב את הקוד ע"פי ה Naming Convention המקובלים של שפת #C .
פירוט תוכלו למצוא בקישור הבא:
https://msdn.microsoft.com/en-us/library/ms229002(v=vs.110).aspx
עליכם להגיש את כל הקוד של הפרויקט מתועד ע"פ הסטנדרט הבא:
https://msdn.microsoft.com/en-us/library/aa288481%28v=vs.71%29.aspx

צורת ההגשה:

עליכם להגיש קובץ zip שיכיל 3 תיקיות:

1. **תיקיית Src:** אשר תכיל את כל הקוד של ה- Solution הכולל את כל הפרויקטים.
2. **תיקיית exe:** אשר תכיל את קובץ ההרצה (קובץ exe והקבצים הנלווים אליו).
   פשוט להעתיק את תיקיית bin של אחד כל אחד ואחד מהפרויקטים.
3. **תיקייה בשם etc:** אשר תכיל קובץ בשם info.txt אשר יכיל את שמות המגישים, ת"ז, קבוצת תרגול של כל אחד. בנוסף היא תכיל את ה- log של העבודה מול ה- git. **(הגשה ללא תעודת זהות תוביל להורדה אוטומטית של 10 נקודות!)**

כל קבצי ההרצה יורצו על מערכת windows עם net framework. חדש אשר אמור לתמוך בכל הגרסאות אחורה. מחובתכם לוודא שכל הקבצים שהגשתם תקינים, ובפרט שהגשתם את **כל הקבצים** הנדרשים. מומלץ לוודא שהקבצים שלכם רצים תקין במחשב שונה משלכם טרם ההגשה.

לא ניתן יהיה להגיש קבצים מחדש או רטרואקטיבית או תיקונים או שינויים לקבצים לאחר מועד ההגשה.
חובה להגיש אך ורק לקבוצת התרגיל שאתם רשומים אליה.

את התרגיל יש להגיש עד השעה 22:00 דרך המודל.

בפועל מערכת ההגשה תאפשר הגשה עד 23:59 של היום הנקוב, אך הגשה לאחר 22:00 היא "על אחריותכם" בלבד. **לא ניתן להגיש באיחור כלל**.

בהצלחה

דור ואלי