פרק 11 – מחלקות ועצמים: הרחבה והעמקה

בפרקים הקודמים ראינו כי בשפת C# ניתן להשתמש בטיפוסים מורכבים הנקראים מחלקות (**classes**). משתנים מטיפוסים אלה נקראים עצמים (**objects**). בפרק 9 ראינו שימוש בעצמים מהמחלקה **string** המוגדרת בשפת C#.

בפרק זה נלמד כיצד להגדיל את אוצר הטיפוסים שעומדים לרשותנו, כלומר כיצד להגדיר מחלקות חדשות בעצמנו ולהשתמש בעצמים ממחלקות אלו.

11.1 מחלקה - הגדרה ושימוש

בעיה 1

מטרת הבעיה ופתרונה: הכרת המושגים מחלקה, תכונות, פעולות ועצמים. הגדרת מחלקה בסיסית, יצירת עצם ושימוש בו.

> אפשר לייצג זמן באמצעות שעות ודקות. כתבו תוכנית הקולטת מהמשתמש זמן בשני ערכים: שעה (מספר בין 0 ל-23) ודקה (מספר בין 0 ל-59) ומציגה את הזמן בפורמט סטנדרטי, למשל עבור השעה אחת עשרה ועשרה הפלט יהיה: 11:10.

השאלה עוסקת בזמן. אפשר להסתכל על כל זמן כעל ישות המתאפיינת באמצעות שני נתונים – אחד מייצג את השעה והשני את הדקה. אם כך, ניתן להגדיר טיפוס חדש (מחלקה) בשם "זמן" אשר יאגד בתוכו את תכונות הזמן. לאחר שנגדיר את הטיפוס "זמן" נוכל ליצור עצמים (הנקראים גם מופעים) המייצגים זמנים שונים. למופעים של הטיפוס החדש, כלומר לכל עצם מהטיפוס זמן, יהיו התכונות שעה ודקה אשר מתארות זמן מסוים.

בנוסף להגדרת התכונות (**attributes**), אנו צריכים להגדיר את הפעולות (**methods**) שנרצה להפעיל על עצמים מסוג זמן. מתוך תיאור הבעיה אנו מבינים שאנחנו זקוקים לפעולה המעדכנת את הזמן לפי ערכים המתקבלים מהמשתמש, כלומר אנחנו נרצה לעדכן את ערכי תכונות הזמן כך שכל עצם מסוג זמן ייצג זמן מסוים כלשהו כגון: שמונה ועשרה, שתים עשרה ארבעים וחמש וכו'. פעולה נוספת המתבקשת מתיאור הבעיה היא פעולה המחזירה את הייצוג הסטנדרטי של הזמן. כלומר בהינתן עצם מסוג זמן נרצה לקבל ממנו מחרוזת בפורמט המבוקש, למשל 8:10 או 12:45.

לצורך פתרון הבעיה, נגדיר תחילה את הטיפוס החדש, את המחלקה *זמן* ולאחר מכן נמשיך בשלבי פיתוח הפתרון.

הגדרת המחלקה *זמן*

כפי שראינו, לעצמים יש תכונות המאפיינות אותם ופעולות השייכות להם. בבואנו להגדיר מחלקה חדשה עלינו להגדיר את התכונות ואת הפעולות שתהיינה לעצמים מהמחלקה.

הגדרת התכונות

כדי להגדיר את התכונות של זמן, עלינו לקבוע מהו הטיפוס המתאים לכל תכונה. שעה היא תכונה שערכה יכול להיות מספר שלם בין 0 ל-23, ודקה היא תכונה שערכה יכול להיות מספר שלם בין 0 ל-59. לכן נגדיר אותן מטיפוס שלם.

בהתאם להחלטה על הטיפוסים נבחר משתנים לייצוג התכונות.

בחירת משתנים לייצוג התכונות

- **hour** – משתנה מטיפוס שלם המייצג את מרכיב השעות בזמן.
- **minute** – משתנה מטיפוס שלם המייצג את מרכיב הדקות בזמן.

הגדרת הפעולות

כאמור, על פי הגדרת הבעיה, אפשר לבצע על זמן את הפעולות הבאות.

- **עדכון הזמן** – פעולה זו מקבלת שני פרמטרים: אחד מייצג את השעות והשני את הדקות, והיא מעדכנת את רכיבי הזמן בהתאם.
- **החזרת הייצוג הסטנדרטי של הזמן** – פעולה זו מחזירה מחרוזת המייצגת את הזמן. אפשר לבנות את המחרוזת באמצעות שרשור התכונות עם סימן של נקודתיים ביניהן:

```
hour + ":" + minute
```

הגדרנו את תכונות הזמן ואת פעולותיו כפי שנובעות מהגדרת הבעיה. נראה כעת כיצד לממש הגדרות אלו בשפת C#.

מימוש המחלקה

כזכור מהאופן שאנו מגדירים את המחלקה הראשית, הגדרת מחלקה בשפת C# מתחילה תמיד במילה השמורה **class** ולאחר מכן נכתב שם המחלקה. בשפת C# מקובל להתחיל שם מחלקה באות גדולה. פרטי ההגדרה מופיעים בתוך סוגריים מסולסלים.

אם כן, כך נראית מסגרת הגדרת מחלקה בשם `Time`:

```
public class Time
{
        // גוף המחלקה //
} // class Time
```

מאפיין הגישה **public** מציין שהמחלקה היא ציבורית ולכן ניתנת לשימוש מכל מחלקה אחרת. וכך מתוך הפעולה הראשית (הנמצאת במחלקה אחרת) נוכל ליצור עצמים מסוג "זמן" ולהשתמש בהם.

בתוך תחום ההגדרה, כלומר בתוך הסוגריים המסולסלים, נפרט את המרכיבים השונים: תכונות ופעולות.

מימוש התכונות:

```
public class Time
{
    private int hour;           //מכיל את השעות
    private int minute;         //מכיל את הדקות
} // class Time
```

הגדרה של תכונה נעשית בדומה להצהרה על משתנה רגיל: שם הטיפוס (למשל **int**) ואחריו שם התכונה (למשל, `hour`). המילה השמורה **private**, המופיעה בתחילת כל הצהרה מציינת כי התכונה היא פרטית, ואין אליה גישה ישירה מחוץ למחלקה. לכן מתוך הפעולה הראשית (המכונה Main) לא נוכל להתייחס ישירות לתכונת השעות ולתכונת הדקות של עצם מסוג *זמן*. ניסיון להתייחסות ישירה כזאת גורם לשגיאת הידור. לעומת זאת, **פעולות** של זמן המוגדרות בתוך תחום המחלקה *זמן* יכולות להתייחס **לתכונות** של הזמן כאל משתנים לכל דבר. מכך נובע שכל התייחסות לתכונות של עצם נעשית אך ורק דרך הפעולות של אותו העצם.

אתחול התכונות:

בדומה למשתנה רגיל, אפשר גם לספק ערכים התחלתיים לתכונות, למשל:

```
private int hour = 8;
private int minute = 0;
```

במקרה זה ערכו ההתחלתי של עצם מסוג זמן יהיה השעה שמונה. לאחר מכן נוכל להשתמש בפעולת העדכון כדי לשנות את הזמן. במקרה שלא מאתחלים את התכונות במפורש, שפת C# מאתחלת תכונות מטיפוס מספרי בערך אפס, תווים מאותחלים בתו מיוחד המייצג תו ריק, תכונות בוליאניות בערך **false** ומערכים ועצמים בערך מיוחד **null**.

ההצהרות על התכונות יכולות להופיע בכל מקום בתוך תחום המחלקה. אנו ננהג לרשום את ההצהרות בראש המחלקה ואחריהן נרשום את הפעולות.

מימוש הפעולות:

נפנה כעת למימוש הפעולות. נתחיל בפעולה לעדכון הזמן המקבלת את השעה ואת הדקה של עצם מסוים ומעדכנת את תכונותיו. מימוש הפעולה `SetTime` בשפת C# נעשה כך:

```
public void SetTime(int h, int m)
{
    hour = h;
    minute = m;
}
```

נסביר את מרכיבי הפעולה.

השורה הראשונה היא **כותרת הפעולה**. היא מהווה למעשה את תעודת הזהות של הפעולה, ומציגה לפני המשתמש את כל המידע הדרוש לצורך שימוש בפעולה. למעשה, כבר ראינו כותרות של פעולות: בפרק 9, הכרנו כמה פעולות של עצמים מסוג **`string`**, וכדי שנדע כיצד להשתמש בפעולות אלו הצגנו בטבלה עבור כל פעולה את שורת הכותרת שלה. שורה זו גילתה לנו פרטים חשובים: את שם הפעולה, אילו פרמטרים היא מצפה לקבל ואיזה סוג ערך היא מחזירה. כותרת הפעולה `SetTime` מודיעה כי היא מצפה לקבל שני פרמטרים מסוג **`int`** הנקראים `h` ו-`m` (כפי שמציינים הסוגריים), ושהיא אינה מחזירה ערך כלשהו (כך מציינת המילה **`void`**). המילה השמורה **`public`** בתחילת שורת הכותרת מציינת כי הפעולה `SetTime` מוגדרת כפעולה ציבורית של עצם מסוג *זמן*. פעולה ציבורית של עצם היא פעולה שאפשר להפעיל על העצם, ולזמן אותה מכל מקום בתוכנית. כך נוכל מתוך הפעולה הראשית להפעיל את הפעולה `SetTime` על עצמים מסוג *זמן*, כפי שנדגים מיד.

גוף הפעולה תחום כרגיל בתוך סוגריים מסולסלים. בתוך הסוגריים אנו כותבים את ההוראות המממשות את עדכון תכונות הזמן. שימו לב שהפעולה `SetTime` משתמשת בתכונות הפרטיות שהצהרנו עליהן, ושרק דרך פעולות שהגדרנו במחלקה ניתן לגשת לתכונותיה המוגדרות כ-**`private`**.

באופן דומה נגדיר את פעולת החזרת הייצוג הסטנדרטי של הזמן:

```
public string GetTime()
{
    return hour + ":" + minute;
}
```

כותרת הפעולה מציינת ששם הפעולה הוא `GetTime`, הפעולה לא מקבלת פרמטרים (כפי שמציינים הסוגריים הריקים), אך היא מחזירה ערך מסוג **`string`**.

גוף הפעולה כולל משפט **return** שתפקידו לסיים את ביצוע הפעולה ולהחזיר לנקודה שממנה הופעלה הפעולה את הערך הרשום בביטוי המופיע אחרי המילה **return**. במקרה זה תוחזר המחרוזת `hour + ":" + minute` המייצגת את הזמן. טיפוס הביטוי המוחזר בהוראה **return** חייב להיות זהה לטיפוס המופיע בכותרת הפעולה (במקרה זה **string**). במקרה של פעולות שלא מחזירות ערך כמו הפעולה `SetTime` אין צורך במשפט **return**, והחֲזרה מהפעולה מתבצעת עם ההגעה לסוף תחום הפעולה.

הנה ההגדרה המלאה של המחלקה *זמן* :

```
/*
  מחלקת זמן
*/
public class Time
{
    // הגדרת התכונות
    private int hour = 0;      // שעות
    private int minute = 0;    // דקות
    // פעולה לעדכון הזמן
    public void SetTime(int h, int m)
    {
        hour = h;
        minute = m;
    }
    //פעולה להחזרת הייצוג הסטנדרטי של הזמן//
    public string GetTime()
    {
        return hour + ":" + minute;
    }
} // class Time
```

הגדרת הפעולה הראשית

כתבנו מחלקה המגדירה את הטיפוס זמן וכעת עלינו לכתוב את התוכנית לפתרון הבעיה. התוכנית תיכתב במחלקה נפרדת בתוך הפעולה הראשית (המכונה Main) כפי שאנו יודעים.

פירוק הבעיה לתת-משימות

תיאור הבעיה מוביל לפירוק הבא לתת-משימות :

1. יצירת עצם מסוג זמן
2. קליטת נתוני זמן (שעה ודקה) ועדכון העצם מסוג זמן
3. הצגת הייצוג הסטנדרטי של הזמן

פתרון המשימה יהיה דומה לתוכניות שכתבנו בפרקים הקודמים. אבל במקום להשתמש רק בטיפוסים המוגדרים מראש בשפה, נשתמש גם במחלקה החדשה שבנינו, המחלקה *זמן*. אם כך, נעבור לשלב בחירת המשתנים :

בחירת משתנים

tm – עצם מהמחלקה Time.

h – השעה כפי שניתן על-ידי המשתמש.

m – הדקה כפי שניתן על-ידי המשתמש.

מימוש

כמו שלמדנו בפרקים קודמים, בניגוד למשתנים מטיפוסים פשוטים הנוצרים באופן אוטומטי עם ההצהרה עליהם, כאשר אנו משתמשים בעצמים עלינו קודם כל ליצור אותם. תהליך יצירת עצם כולל הקצאת מקום בזיכרון עבור העצם ואתחול של תכונותיו. יצירת עצם נעשית באמצעות ההוראה **new**, למשל כך:

```
Time tm;
tm = new Time();
```

אחרי שיצרנו את העצם tm, נוכל להפעיל עליו את הפעולות שהוגדרו עבורו. הפעלת פעולה מתבצעת באמצעות סימון הנקודה, למשל, המשפט הבא מעדכן את הזמן בעצם tm:

```
tm.SetTime(h,m);
```

שימו ♥: לא ניתן להתייחס באמצעות סימון הנקודה, לתכונותיו של העצם tm מתוך הפעולה הראשית, כי אלו הוגדרו כפרטיות (**private**). לכן הביטויים tm.hour ו-tm.minute אינם חוקיים.

הנה המימוש המלא של המחלקה הראשית:

```
/*
        המחלקה הראשית המשתמשת במחלקה זמן
*/
using System;
public class UseTime
{
    public static void Main()
    {
        // הגדרת משתנים
        Time tm;
        int h;
        int m;
        //יצירת עצם מסוג זמן
        tm = new Time();
        // קלט
        Console.Write("Enter the hour (0..23): ");
        h = int.Parse(Console.ReadLine());
        Console.Write("Enter the minute (0..59): ");
        m = int.Parse(Console.ReadLine());
        // עדכון הזמן והצגת פלט
        tm.SetTime(h,m);
        Console.WriteLine("The time is: {0}", tm.GetTime());
    } // Main
} // class UseTime
```

סוף פתרון בעיה 1

נסכם את הנלמד בפתרון בעיה 1:

לצורך פתרון הבעיה הגדרנו טיפוס חדש, את המחלקה זמן. לכן, שלא כמו התוכניות שכתבנו עד כה שכללו רק מחלקה אחת והיא המחלקה הראשית, בתוכנית זו יש שתי מחלקות: האחת מגדירה את הטיפוס החדש Time והשנייה היא המחלקה הראשית המגדירה את הפעולה Main שניתן להריץ במחשב. הפעולה Main יוצרת עצם מסוג Time ומפעילה את פעולותיו.

שימו ♥: נהוג בשפת C# לשים כל מחלקה בקובץ נפרד אף על פי שניתן לשים יותר ממחלקה אחת בקובץ.

- **הגדרת מחלקה** היא למעשה הגדרה של טיפוס חדש. הגדרת מחלקה כוללת הגדרה של תכונות ושל פעולות עבור עצמים של המחלקה, כלומר עבור משתנים מהטיפוס החדש.
- **תכונות של עצם** הן משתנים מטיפוסים כלשהם, המאפיינים את העצם.
- **פעולות של עצם** אף הן משויכות לעצם, והן פועלות בדרך כלל על תכונות העצם.
- כדי לא לחשוף את אופן המימוש הפנימי של עצמים, נקפיד להגדיר את התכונות כ**פרטיות**. כלומר מחוץ למחלקה לא ניתן להתייחס אליהן ישירות.
- פעולות יוגדרו בדרך כלל כ**ציבוריות**.

- **הגדרת מחלקה בשפת C#** נכתבת באופן הבא:

```
public class שם המחלקה
{
    // הגדרת התכונות
    .
    .
    // הגדרת הפעולות
    .
    .
}
```

- המילה השמורה **public** מציינת שהמחלקה פתוחה לשימוש ציבורי. כלומר שמחלקות אחרות יכולות ליצור עצמים מטיפוס המחלקה ולהפעיל את הפעולות שלהם.
- המילה השמורה **class** מציינת את הגדרתה של מחלקה חדשה בשפה. לאחר ציון המילה **class** נציין את שם המחלקה. מקובל להתחיל שם מחלקה באות גדולה. אם שם המחלקה מורכב מכמה מילים, הן נכתבות צמודות כאשר כל מילה מתחילה באות גדולה ושאר האותיות הן קטנות.
- **הגדרת תכונות בשפת C#** נכתבת בדומה להצהרה על משתנים. הצהרה של תכונה של עצם כוללת את הרשאת הגישה לתכונה, את הטיפוס של התכונה ולאחר מכן את שמה. הרשאת גישה פרטית לתכונות מוגדרת באמצעות המילה השמורה **private**, למשל:

```
private int hour;
```

- אפשר לאתחל תכונה בעת הצהרתה, למשל:

```
private int hour = 0;
```

במקרה שלא מאתחלים את התכונות במפורש, שפת C# מאתחלת תכונות מטיפוס מספרי בערך אפס, תווים מאותחלים בתו מיוחד המייצג תו ריק, תכונות בוליאניות בערך **false** ומערכים ועצמים בערך מיוחד **null**.

- ניתן לגשת לתכונות פרטיות של עצם רק מתוך פעולות העצם – פעולות המוגדרות באותה המחלקה.

- ## הגדרת פעולה בשפת C# נכתבת באופן הבא:

```
public טיפוס-הערך-המוחזר שם-הפעולה(רשימת פרמטרים)
{
    גוף הפעולה
}
```

- ## כותרת הפעולה

כותרת הפעולה נותנת למשתמש את כל המידע הדרוש לצורך שימוש בפעולה. כותרת הפעולה כוללת את הרכיבים הבאים:

הרשאת גישה לפעולה

רכיב זה קובע מי רשאי להפעיל את הפעולה של העצם. משמעות המילה **public** היא כי הפעולה יכולה להיות מופעלת מכל מחלקה אחרת בתוכנית.

טיפוס הערך המוחזר

כאשר הפעולה איננה מחזירה דבר נכתוב **void** כטיפוס הערך המוחזר. בכל מקרה אחר, נכתוב את שם הטיפוס, והוא יכול להיות טיפוס פשוט בשפה (כמו **int**), מערך (כמו **int[]**) או שם מחלקה (כמו **string**).

שם הפעולה

שם הפעולה הוא שם המתאר את הפעולה המתבצעת. נהוג ששם הפעולה יתחיל באות גדולה. אם השם מורכב מכמה מילים, הן נכתבות צמודות כאשר כל מילה מתחילה באות גדולה ושאר האותיות הן קטנות.

הפרמטרים

כאשר פעולה לא מגדירה פרמטרים הסוגריים נותרים ריקים. בכל מקרה אחר הרשימה מורכבת מזוגות המופרדים בפסיקים. כל זוג מתאר פרמטר אחד וכולל את טיפוס הפרמטר ואת שמו.

- ## גוף הפעולה

את גוף הפעולה יש לתחום בסוגריים מסולסלים (הסימנים {...}).

בגוף הפעולה נכתוב את רצף ההוראות שמבצע את מטרת הפעולה, כלומר מיישם את האלגוריתם של הפעולה. בגוף הפעולה ניתן להצהיר על משתנים ולהשתמש בכל ההוראות הקיימות בשפת C#. כגון משפטי תנאי, לולאות, השמות ועוד. במקרה שמצהירים על משתנים, משתנים אלה מוּכָּרִים רק בגוף הפעולה, הם נהרסים עם סיום ביצוע הפעולה ולא ניתן להשתמש בהם מחוץ לה.

שימו ♥: בגוף הפעולה אפשר להתייחס ישירות לכל התכונות של העצם שמופעלת עליו הפעולה. אמנם התכונות הן פרטיות, כלומר מוחבאות משאר חלקי התוכנית, אולם הן ידועות וגלויות לכל פעולות העצם המוגדרות במחלקה. לכן מתוך הפעולות מותרת התייחסות ישירה לתכונות.

- ## הוראת חזרה return

כאשר ביצועה של ההוראה האחרונה בפעולה מסתיים, הפעולה כולה מסתיימת. אבל ניתן גם לגרום במפורש לסיום הביצוע באמצעות הוראת החזרה **return**. הוראה זו גורמת ליציאה מיידית מהפעולה. כאשר הפעולה אמורה להחזיר ערך, הוראת **return** כוללת גם ביטוי שיוחזר עם סיום הביצוע. טיפוס הביטוי המוחזר מהפעולה חייב להיות תואם לטיפוס הערך המוחזר שמוגדר בכותרת הפעולה.

שימו ♥: למען קריאות המחלקה חשוב מאוד לצרף להגדרה הערות המתארות את המחלקה, את תכונותיה ואת פעולותיה.

- כדי **להשתמש במחלקה** ניצור ממנה עצם במחלקה אחרת (למשל במחלקה הראשית).
- כדי **ליצור עצם** נצהיר עליו ונקצה לו מקום בזיכרון באמצעות ההוראה **new**, למשל:

```
Time tm;
tm = new Time();
```

- כדי **להפעיל את פעולות העצם** (לזמן את פעולות העצם) נשתמש בסימון הנקודה, למשל:

```
tm.SetTime(h,m);
```

יש לשים לב להתאמה בין אופן השימוש בפעולה של העצם לבין כותרת הפעולה: אם הפעולה מצפה לקבל פרמטרים יש להקפיד על סדר הפרמטרים כפי שמופיע בכותרת הפעולה, ועל התאמה של טיפוס הפרמטרים המועברים לטיפוסים המפורטים בכותרת הפעולה; אם הפעולה מחזירה ערך יש להקפיד על התאמה של טיפוס הערך המוחזר לטיפוס הביטוי שמשולב בו הערך.

שאלה 11.1

א. הוסיפו למחלקה Time פעולה בשם Increment המקדמת את הזמן בדקה. למשל לאחר הפעולה Increment הזמן 10:52 ישתנה ל- 10:53, והזמן 23:59 ישתנה ל- 0:0.

ב. כתבו פעולה ראשית הקולטת מהמשתמש נתוני זמן (שעה ודקה) ומספר שלם k. התוכנית תקדם את הזמן ב-k דקות (באמצעות הפעולה שהוגדרה בסעיף הקודם) ותציג את השעה שהתקבלה.

בעיה 2

מטרת הבעיה ופתרונה: היכרות עם פעולה בונה (**constructor**).

בשיעורי הנדסה בכיתה ד' המורה לימדה כיצד לחשב שטח והיקף של מלבן. כעת רוצה המורה לבחון את תלמידיה. המבחן כולל 20 שאלות שבכל אחת מהן התלמיד נדרש לחשב את שטחם ואת היקפם של מלבנים אשר אורכיהם ורוחביהם נתונים. פתחו וממשו תוכנית שתעזור למורה לחשב את התשובות למבחן. התוכנית תקלוט עבור כל שאלה את האורך ואת הרוחב של המלבן ותציג את שטחו ואת היקפו.

מתוך תיאור הבעיה ניתן לזהות שהשאלה עוסקת במלבנים. אפשר להסתכל על כל מלבן כעל ישות המתאפיינת בשני נתונים – אורך ורוחב, ומתאפיינת בשתי פעולות – אחת לחישוב השטח והאחרת לחישוב ההיקף. אם כך, ניתן להגדיר מחלקה בשם מלבן אשר יאגד בתוכו את תכונות המלבן ואת הפעולות של המלבן. לאחר שנגדיר את הטיפוס מלבן נוכל ליצור מלבנים שונים, כלומר עצמים, שכל אחד מהם ייצג מלבן אחר. לכל מופע (עצם) של הטיפוס מלבן, יהיו גם התכונות אורך ורוחב אשר מתארות את המלבן וגם הפעולות שטח והיקף שניתן להפעיל על המלבן.

לפתרון הבעיה נגדיר תחילה את המחלקה *מלבן* ולאחר מכן נמשיך בשלבי פיתוח הפתרון.

הגדרת המחלקה *מלבן*

הגדרת התכונות

אורך ורוחב הן יחידות מידה. לכן נגדיר אותן מטיפוס ממשי. בהתאם לכך נבחר את המשתנים הבאים לתיאור התכונות:

- **length** – משתנה מטיפוס ממשי המייצג את אורך המלבן.
- **width** – משתנה מטיפוס ממשי המייצג את רוחב המלבן.

הגדרת הפעולות

על פי הגדרת הבעיה, מלבן יכול לבצע את הפעולות הבאות:

- **חישוב שטח** – פעולה זו מחשבת ומחזירה את שטח המלבן. בפעולה זו מוגדרת משימה אלגוריתמית, עם נקודת מוצא (אורך ורוחב המלבן) ופלט מבוקש (שטח). ניתן לחשב את שטח המלבן באמצעות הביטוי הבא:

width * length

- **חישוב היקף** – פעולה זו מחשבת ומחזירה את היקף המלבן. הנה הביטוי לחישוב ההיקף:

2 * width + 2 * length

הגדרנו את הפעולות שטח והיקף. פעולות אלו משתמשות בתכונות אורך המלבן ורוחב המלבן. כדי לאתחל את תכונות המלבן בגדלים של מלבן מסוים ניתן להגדיר פעולה מיוחדת המיועדת לאתחול תכונות העצם. פעולה זו נקראת **פעולה בונה** והיא מתבצעת מיד עם יצירת עצם מסוג מלבן. אם כך, נוסיף את הפעולה הבאה:

- **פעולה בונה -** פעולה זו תקבל כפרמטרים את אורך המלבן ואת רוחבו ותאתחל את התכונות בהתאם.

הגדרנו את תכונות המלבן ואת פעולותיו כפי שנובעות מהגדרת הבעיה. כעת נממש את ההגדרות האלו בשפת #C.

מימוש המחלקה

נגדיר מחלקה בשם `Rectangle`. בתוך תחום המחלקה נפרט את המרכיבים השונים:

התכונות אורך ורוחב:

```
private double length; //מכיל את אורך המלבן

private double width; // מכיל את רוחב המלבן
```

הפעולות שטח והיקף:

```
public double Area()
{
   return width * length;
}
public double Perimeter()
{
   return 2 * width + 2 * length;
}
```

הפעולה הבונה:

כאמור הפעולה הבונה היא פעולה מיוחדת המיועדת לאתחול תכונות העצם. פעולה זו תופעל בעת יצירת המופע והיא מחזירה הפניה לעצם שנוצר. הגדרת הפעולה הבונה דומה להגדרת פעולה רגילה כמו חישוב שטח וחישוב היקף שהוצגו לעיל, פרט לכך ששמה זהה לשם המחלקה ולא רושמים את טיפוס הערך המוחזר (גם לא **void**). הפעולה הבונה, כמו כל פעולה, יכולה לקבל פרמטרים.

אם כך, נוכל להגדיר עבור המחלקה מלבן פעולה בונה ששמה `Rectangle`, והיא תקבל את האורך ואת הרוחב של המלבן. נשתמש בה כדי לאתחל את תכונותיו של עצם מסוג מלבן.

הנה פעולה בונה עבור עצמים מסוג *מלבן* :

```
public Rectangle(double aLength, double aWidth)
{
    length = aLength;
    width = aWidth;
}
```

שימו ♥ : השם של הפעולה הבונה חייב להיות זהה לשם המחלקה, ובכותרת שלה אין טיפוס לערך מוחזר.

יש מקרים שבהם משתמשים בשמות פרמטרים זהים לשמות התכונות. הדבר נפוץ במיוחד בפעולות שהפרמטרים בהן מתייחסים בצורה ישירה לתכונות. למשל ניתן לכתוב את הכותרת של הפעולה הבונה באופן הבא :

```
public Rectangle(double length, double width)
```

נרצה שהפרמטר `length` יאתחל את התכונה `length`, ושהפרמטר `width` יאתחל את התכונה `width`. מכיוון ששמות הפרמטרים זהים לשמות התכונות, אנו צריכים דרך להבחין ביניהם. אם בתוך הפעולה נתייחס ישירות למזהים `length` ו-`width` אז נתייחס למעשה לפרמטרים `length` ו-`width` ולא לתכונות באותם השמות. אם כך, כיצד נתייחס לתכונות?

בשפת C# קיימת בתוך כל פעולה של עצם הפניה בשם **this** המפנה לעצם עצמו – העצם שהפעולה שלו הופעלה. כלומר אם נרשום בתוך הפעולה את הביטוי **this**`.length` אז נתייחס לתכונה `length` של העצם שאנו יוצרים ולא לפרמטר `length`.

אם כך, נוכל לכתוב את הפעולה הבונה כך :

```
public Rectangle(double length, double width)
{
    this.length = length;
    this.width = width;
}
```

באופן כללי, כאשר שם של פרמטר זהה לשם של תכונה, נשתמש בקידומת **this** כדי לציין את התכונה.

הנה ההגדרה המלאה של המחלקה *מלבן* :

```
/*
 מחלקת מלבן
*/
public class Rectangle
{
    // הגדרת התכונות
    private double width;      // רוחב
    private double length;     // אורך
    //פעולה בונה
    public Rectangle(double length, double width)
    {
        this.length = length;
        this.width = width;
    }
```

```
    // פעולת שטח
    public double Area()
    {
        return width * length;
    }
    //פעולת היקף
     public double Perimeter()
    {
        return 2 * width + 2 * length;
    }
} // class Rectangle
```

הגדרת הפעולה הראשית

כתבנו מחלקה המגדירה את הטיפוס מלבן וכעת עלינו לכתוב את הפעולה הראשית לפתרון הבעיה.

פירוק הבעיה לתת-משימות

תיאור הבעיה מוביל לפירוק הבא לתת-משימות:
עבור כל שאלה מ-20 שאלות המבחן:

1. קליטת אורך ורוחב המלבן ויצירת עצם מסוג מלבן
2. חישוב שטח המלבן (באמצעות פעולת השטח) והצגתו
3. חישוב היקף המלבן (באמצעות פעולת ההיקף) והצגתו

בחירת משתנים

rec – עצם מהמחלקה Rectangle. מייצג את המלבן עבור כל שאלה.
length – אורך המלבן כפי שניתן על-ידי המשתמש.
width – רוחב המלבן כפי שניתן על-ידי המשתמש.

האלגוריתם

1. בצע 20 פעמים:
   1.1 קלוט את אורך המלבן ב-length
   1.2 קלוט את רוחב המלבן ב-width
   1.3 צור עצם מסוג Rectangle בשם rec ואתחל אותו באורך וברוחב שנקלטו מהמשתמש.
   1.4 הפעל את פעולת שטח המלבן על rec והצג את התוצאה
   1.5 הפעל את פעולת היקף המלבן על rec והצג את התוצאה

מימוש

כפי שראינו בבעיה הקודמת, כאשר אנו משתמשים בעצם עלינו קודם כל להקצות עבורו מקום חדש בזיכרון ולאתחל את תכונותיו. הקצאת המקום מתבצעת באמצעות ההוראה **new** והאתחול מתבצע באמצעות קריאה לפעולה הבונה, למשל כך:

```
Rectangle rec;
rec = new Rectangle (5,2.5);
```

ואכן, עם הקצאת המקום לעצם `rec` מופעלת הפעולה הבונה ששמה `Rectangle`. פעולה זו מבצעת את האתחולים הדרושים לצורך תחילת עבודה תקינה עם העצם שנוצר. משום כך היא נקראת **פעולה בונה**.

הנה המימוש המלא של המחלקה הראשית:

```
/*
        המחלקה הראשית המשתמשת במחלקה מלבן
*/
using System;
public class TestSolver
{
    public static void Main()
    {
        // הגדרת משתנים
        Rectangle rec;
        double length;
        double width;
        for (int i = 0; i < 20; i++)
        {
            //  קלט
            Console.Write("Enter the rectangle length: ");
            length = double.Parse(Console.ReadLine());
            Console.Write("Enter the rectangle width: ");
            width = double.Parse(Console.ReadLine());
            //הקצאת מלבן
            rec = new Rectangle(length,width);
            // ביצוע חישובים והצגת פלט //
            Console.WriteLine("The rectangle area is: {0}",
                                                      rec.Area());
            Console.WriteLine("The rectangle perimeter is: {0}",
                                                 rec.Perimeter());
        } // for
    } // Main
} // class TestSolver
```

סוף פתרון בעיה 2

בפתרון בעיה זו הכרנו גם את הפעולה הבונה.

- **הפעולה הבונה** של עצם מופעלת מיד אחרי שההוראה **new** מקצה עבורו שטח בזיכרון.
- **שמה של פעולה בונה** הוא תמיד כשם המחלקה.
- כמו כל פעולה, גם פעולה בונה יכולה לקבל פרמטרים. **פרמטרים** אלה משמשים לאתחול תכונות העצם.
- במקרים ששם של פרמטר זהה לשם של תכונה נוכל להתייחס לתכונה באמצעות הקידומת **this** המציינת התייחסות לתכונות של העצם הנוכחי – העצם שמופעלת עליו הפעולה. אפשר להשתמש ב-**this** מתוך כל פעולה המוגדרת בעצם ולא רק מתוך הפעולה הבונה.
- אם איננו מגדירים פעולה בונה כלשהי במחלקה, שפת C# מספקת פעולה בונה ריקה חסרת פרמטרים. פעולה בונה זו היא ברירת המחדל במחלקה שלא הגדרנו עבורה פעולה בונה. ברגע שנגדיר פעולה בונה במחלקה, היא תחליף את הפעולה הבונה שסופקה כברירת מחדל.

שאלה 11.2

הגדירו וממשו את המחלקות הבאות בשפת C# :

המחלקה	תכונות	פעולות
חתול	- שם - סוג - אוהב או לא אוהב מים - אוהב או לא אוהב ליילל	- פעולה בונה המאתחלת את תכונות העצם. - האם צמא : עבור חתול שאוהב מים הפעולה תחזיר **true** אחרת יוחזר **false**. - יללה : עבור חתול שאוהב ליילל הפעולה תחזיר את המחרוזת "מיאו מיאו מיאו". עבור חתול שאינו אוהב ליילל הפעולה תחזיר את המחרוזת "מיאו". - בירור פרטי החתול : הפעולה תחזיר את המחרוזת : שמי <שם החתול> והסוג שלי הוא <סוג החתול>.
ילד	- שם - גיל	- פעולה בונה המאתחלת את תכונות העצם. - בירור פרטים אישיים : הפעולה תחזיר את המחרוזת : שמי <שם הילד> וגילי הוא <גיל הילד>. - יום הולדת : הפעולה תקדם את גיל הילד באחד.
מעגל	- רדיוס	- פעולה בונה המאתחלת את תכונות העצם. - שטח : הפעולה תחזיר את שטח המעגל. - היקף : הפעולה תחזיר את היקף המעגל. **הדרכה** : השתמשו בקבוע `Math.PI`

שאלה 11.3

כתבו תוכנית המשתמשת במחלקת החתול שהוגדרה בשאלה הקודמת. התוכנית תקלוט מהמשתמש נתונים עבור חתולים (שם, סוג, האם הוא אוהב מים והאם הוא אוהב ליילל). עבור כל חתול, יוצג כפלט : פרטי החתול (שם וסוג), היללה שלו והאם הוא צמא.

התוכנית תסתיים כאשר המשתמש יקליד מחרוזת ריקה עבור שם החתול.

11.2 פעולות גישה

בעיה 3

מטרת הבעיה ופתרונה : היכרות עם פעולות גישה המחזירות ומעדכנות את ערכי תכונות העצם.

לאמא ארנבת שני ארנבונים : ארני וברני. בכל יום שישי מודדת אמא ארנבת את אורך אוזני גוריה ואת משקלם. עזרו לאמא ארנבת לנהל את מדידותיה. הגדירו וממשו מחלקה בשם "ארנבון" בה יישמרו נתוני כל ארנבון : שמו, אורך אוזניו ומשקלו. שימו לב שאורך אוזני הארנבון אינו יכול לקטון.

כתבו תוכנית שנתית המשמשת למדידת הארנבונים. תחילה התוכנית תיצור את שני הארנבונים ארני וברני, ולאחר מכן היא תקלוט עבור כל שבוע (52 שבועות) את אורך האוזניים ואת המשקל של כל ארנבון. אם אמא ארנבת שגתה בהכנסת הנתונים והכניסה אורך אוזניים קצר יותר מהנוכחי, תציג התוכנית הודעת שגיאה ותבקש להכניס שוב את הנתונים. בסוף השנה תדפיס התוכנית את הנתונים העדכניים של כל ארנבון.

הגדרת המחלקה *ארנבון*

לצורך פתרון הבעיה נזדקק למחלקה המגדירה ארנבון.

הגדרת התכונות

על פי הגדרת הבעיה, לעצם מהמחלקה ארנבון יהיו התכונות: שם ארנבון, אורך אוזניים ומשקל. בהתאם לכך נבחר את המשתנים הבאים לתיאור התכונות:

- **name** – מחרוזת המייצגת את שם הארנבון.
- **earsLength** - מספר מטיפוס שלם המייצג את אורך האוזניים של הארנבון.
- **weight** - מספר מטיפוס שלם המייצג את משקל הארנבון.

הגדרת הפעולות

- **פעולה בונה** – הפעולה הבונה תקבל כפרמטר את שם הארנבון, ותאתחל את תכונת שם הארנבון. מה לגבי אתחול שאר תכונות המחלקה, משקל ואורך אוזניים? בתחילה, אין נתונים עבור משקל ועבור אורך אוזני הארנבון. הנתונים לגבי תכונות אלו יתקבלו רק לאחר המדידה הראשונה. לכן הפעולה הבונה תאתחל את שם הארנבון לפי הפרמטר שהתקבל ותאתחל את שתי התכונות האחרות בערך התחלתי אפס.

פעולות לעדכון תכונות המחלקה – עבור כל ארנבון לאחר כל מדידה ועבור כל תכונה, נרצה לעדכן את ערך התכונה כפי שנמדד. כיצד נעשה זאת? כמובן שלפעולה הראשית המנהלת את המדידות אין גישה ישירה לתכונות של הארנבונים משום שתכונות אלו מוגדרות כפרטיות. אם כך, יש צורך בפעולות גישה המעדכנות ערך של תכונה, כלומר בפעולה עבור כל תכונה שנרצה לעדכן:

- **SetWeight** – הפעולה תקבל כפרמטר את משקלו של הארנבון כפי שנמדד ותעדכן את משקלו. הפעולה איננה מחזירה ערך.
- **SetEarsLength** – הפעולה תקבל כפרמטר את אורך אוזניו של הארנבון כפי שנמדד, ותעדכן אותו. שימו לב, לפי הגדרת הבעיה אורך אוזני הארנבון אינו יכול לקטון, כלומר עלינו לעדכן את התכונה אורך אוזני הארנבון רק אם האורך שהתקבל כפרמטר גדול יותר מאורך האוזניים הנוכחי של הארנבון. לפי הגדרת הבעיה, אם אמא ארנבת שגתה בהכנסת הנתונים והכניסה אורך אוזניים קצר יותר, תבקש ממנה הפעולה הראשית להכניס את הנתון שוב. כיצד תדע הפעולה הראשית שהאורך שהתקבל קצר מהאורך השמור? נשתמש בצינור הגישה גם בכיוון השני, הפעולה תחזיר ערך בוליאני המציין אם העדכון התבצע או לא.

פעולות גישה המחזירות תכונות של העצם – בסוף השנה תציג הפעולה הראשית עבור כל ארנבון את אורך אוזניו ואת גובהו. מכיוון שלפעולה הראשית המנהלת את המדידות אין גישה ישירה לתכונות של הארנבונים משום שתכונות אלו מוגדרות כפרטיות, נגדיר פעולות גישה לתכונות והן יחזירו את ערכן של התכונות. בדרך כלל פעולות הגישה הן פעולות שאינן מקבלות פרמטרים אלא רק מחזירות ערך מטיפוס התכונה.

- **GetWeight** – הפעולה איננה מקבלת פרמטרים והיא מחזירה את משקלו של הארנבון.
- **GetEarsLength** – הפעולה איננה מקבלת פרמטרים והיא מחזירה את אורך אוזניו של הארנבון.

מימוש המחלקה

```
/*
המחלקה ארנבון
*/
public class Rabbit
{
    private string name;
    private int weight;
    private int earsLength;
    //פעולה בונה מקבלת את שם הארנבון
    public Rabbit(string name)
    {
        this.name = name;
        weight = 0;
        earsLength = 0;
    }
    //פעולה המעדכנת את משקלו של הארנבון
    public void SetWeight(int weight)
    {
        this.weight = weight;
    }
    //פעולה המעדכנת את אורך אוזני הארנבון רק אם האורך שהתקבל גדול
    // מהאורך השמור. מחזירה אמת אם העדכון בוצע, שקר אחרת //
    public bool SetEarsLength(int earsLength)
    {
        if (this.earsLength > earsLength)
            return false;
        this.earsLength = earsLength;
        return true;
    }
    //פעולת גישה המחזירה את משקלו של הארנבון
    public int GetWeight()
    {
        return weight;
    }
    //פעולת גישה המחזירה את אורך אוזניו של הארנבון
    public int GetEarsLength()
    {
        return earsLength;
    }
}//class Rabbit
```

שימו ♥ לאופן שמיושמת הפעולה SetEarsLength: בפעולה זאת תיתכן יציאה (באמצעות המילה **return**) באמצע הפעולה, בלי להגיע עד סופה. אם אורך האוזניים שהתקבל כפרמטר קטן מהאורך השמור, ביצוע הפעולה מסתיים מיד ומחזיר את הערך שקר. לא משנה אילו הוראות כתובות לאחר התנאי, בכל מקרה הן לא יבוצעו. המילה השמורה **return** גורמת ליציאה מיידית מהפעולה. כל ההוראות שנכתבות לאחריה אינן מבוצעות; פעולת העדכון במקרה זה.

הגדרת הפעולה הראשית

הגדרנו מחלקה לייצוג הטיפוס ארנבון, וכעת עלינו להמשיך לפיתוח הפעולה הראשית האחראית על המדידות.

פירוק לתת-משימות

את משימתה של הפעולה הראשית ניתן לפרק לתת-משימות באופן הבא:

1. יצירת הארנבונים: ארני וברני
2. עבור כל אחד מ-52 השבועות בשנה:
   2.1. קליטת משקלו של ארני ועדכונו
   2.2. קליטת אורך אוזניו של ארני ועדכונו, חזרה על כך כל עוד העדכון לא בוצע (פעולת העדכון החזירה **false**)
   2.3. קליטת משקלו של ברני ועדכונו
   2.4. קליטת אורך אוזניו של ברני ועדכונו, חזרה על כך כל עוד העדכון לא בוצע (פעולת העדכון החזירה **false**)
3. הצגה כפלט: בסוף השנה משקלו של ארני הוא <משקלו של ארני> ואורך אוזניו הוא <אורך אוזניו של ארני>
4. הצגה כפלט: בסוף השנה משקלו של ברני הוא <משקלו של ברני> ואורך אוזניו הוא <אורך אוזניו של ברני>

בחירת משתנים

rabit1 – עצם מטיפוס המחלקה ארנבון, מייצג את ארני.

rabit2 – עצם מטיפוס המחלקה ארנבון, מייצג את ברני.

currentWeight – משתנה מטיפוס שלם, משמש לקליטת משקל הארנבונים.

currentEarsLength – משתנה מטיפוס שלם, משמש לקליטת אורך אוזני הארנבונים.

לא נתעכב על כתיבת האלגוריתם המלא הנובע כמעט באופן ישיר מהפירוק לתת-משימות שניתן לעיל.

מימוש

```
/*
     המחלקה הראשית המשמשת למדידת הארנבונים
*/
using System;
public class RabbitsDevelopment
{
    public static void Main()
    {
        Rabbit rabbit1 = new Rabbit("Arni");
        Rabbit rabbit2 = new Rabbit("Barni");
        int currentWeight;
        int currentEarsLength;
        //עבור כל אחד מהשבועות בשנה נעדכן את נתוני הארנבונים
        for(int i = 0; i < 52; i ++)
        {
            Console.Write("Enter Arni's weight: ");
            currentWeight = int.Parse(Console.ReadLine());
            rabbit1.SetWeight(currentWeight);
            Console.Write("Enter Arni's ears length: ");
            currentEarsLength = int.Parse(Console.ReadLine());
            //כל עוד העדכון לא בוצע, נבקש להכניס שוב את האורך
            while (!rabbit1.SetEarsLength(currentEarsLength))
```

```
         {
            Console.Write("ReEnter Arni's ears length:");
            currentEarsLength = int.Parse(Console.ReadLine());
         }
         Console.Write("Enter Barni's weight: ");
         currentWeight = int.Parse(Console.ReadLine());
         rabbit2.SetWeight(currentWeight);
         Console.Write("Enter Barni's ears length:");
         currentEarsLength = int.Parse(Console.ReadLine());
         //כל עוד העדכון לא בוצע, נבקש להכניס שוב את האורך
         while (!rabbit2.SetEarsLength(currentEarsLength))
         {
            Console.Write("ReEnter Barni's ears length:");
            currentEarsLength = int.Parse(Console.ReadLine());
         }
      }
      //פלט
      Console.WriteLine("At the end of the year Arni's weight " +
                            "is: {0}, Arni's ears length is: {1}",
                  rabbit1.GetWeight(), rabbit1.GetEarsLength());
      Console.WriteLine("At the end of the year Barni's weight " +
                           "is: {0}, Barni's ears length is: {1}",
                  rabbit2.GetWeight(), rabbit2.GetEarsLength());
   }//Main
}// RabbitsDevelopment
```

סוף פתרון בעיה 3

פתרון בעיה 3 כלל פעולות גישה לעדכון ערכי תכונות ולהחזרתם. מאחר שתכונות של עצם מוגדרות בדרך כלל כפרטיות, לא ניתן לגשת אליהן ישירות מחוץ למחלקה. אם כך, גישה לתכונות העצם מחוץ למחלקה נעשית באמצעות **פעולות גישה**.

אנו לא חייבים לספק פעולות גישה לכל תכונה. פעולה המעדכנת ערך של תכונה נכתבת רק עבור תכונות שנרצה שהמשתמש בָּעֶצֶם יוכל לעדכן. פעולה כזו מקבלת את הערך החדש של התכונה כפרמטר. היא בדרך כלל לא מחזירה ערך, או שהיא מחזירה משתנה בוליאני המציין אם פעולת העדכון התבצעה בהצלחה. נהוג ששמה של הפעולה יהיה שרשור של המילה Set ולאחריו שם התכונה. למשל עבור התכונה weight שם פעולת העדכון יהיה SetWeight.

פעולת גישה המחזירה ערך של תכונה נכתבת רק עבור תכונות שאת ערכן נרצה שמשתמש חיצוני יוכל לקבל. פעולה כזו אינה מקבלת פרמטרים, אלא מחזירה ערך מטיפוס התכונה. נהוג ששמה של הפעולה יהיה שרשור של המילה Get לשם התכונה. למשל עבור התכונה weight שם פעולת הגישה להחזרת ערך התכונה יהיה GetWeight.

פעולות גישה משמשות צינור גישה בטוח אל תכונות העצם מחוץ למחלקה. פעולת עדכון יכולה לקבוע את חוקיות הערכים המעדכנים את התכונה (בדומה לפעולת העדכון `SetEarsLength`). בנוסף לכך, גם אם המתכנת שינה את ייצוג התכונה (למשל את שמה), עדיין לא ישתנו הערך המוחזר מפעולת הגישה והפרמטר לפעולת העדכון, וכך שאר חלקי התוכנית המשתמשים בפעולות הגישה לא יהיו מושפעים מהשינויים. בשל כך התוכנית כולה תהיה עמידה יותר בפני שינויים.

שאלה 11.4

כתבו מחלקה המגדירה טיפוס קלמר, כמתואר בשתי הטבלאות הבאות:

התכונה	טיפוס	תיאור
מספר העטים	שלם	מספר העטים שנמצאים בקלמר
מספר העפרונות	שלם	מספר העפרונות שנמצאים בקלמר

הפעולה	טיפוס ערך מוחזר	רשימת פרמטרים	תיאור
פעולה בונה		מספר העטים ומספר העפרונות בקלמר	הפעולה מאתחלת את מספר העטים ואת מספר העפרונות בקלמר
הוספת עט	void	אין	פעולה שמגדילה את מספר העטים בקלמר ב-1
הוספת עיפרון	void	אין	פעולה שמגדילה את מספר העפרונות בקלמר ב-1
אובדן עט	void	אין	פעולה שמקטינה את מספר העטים בקלמר ב-1
אובדן עיפרון	void	אין	פעולה שמקטינה את מספר העפרונות בקלמר ב-1
החזרת מספר העטים	מספר שלם	אין	פעולת גישה שמחזירה את מספר העטים בקלמר
החזרת מספר העפרונות	מספר שלם	אין	פעולת גישה שמחזירה את מספר העפרונות בקלמר

כתבו תוכנית שתקלוט את מספר העטים ואת מספר העפרונות בקלמר ותיצור עצם מסוג קלמר. לאחר מכן התוכנית תקלוט מספר המציין כמה אירועים עבר הקלמר, כאשר אירוע יכול להיות הוספה או איבוד של עט או עיפרון. לאחר מכן התוכנית תקלוט תיאור של כל האירועים, כאשר עבור כל אירוע ייקלט קודם כל סוגו כמספר שלם (1 – הוספה, 2 – איבוד), ולאחר מכן ייקלט סוג החפץ המשתתף באירוע, גם כן כמספר שלם (1 – עט, 2 – עפרון).

פלט התוכנית יהיה מספר העטים ומספר העפרונות בקלמר לאחר כל האירועים שעברו עליו.

למשל, אם נתוני הקלמר ההתחלתיים הם 10 4 (כלומר, 4 עטים ו-10 עפרונות), מספר האירועים הוא 3, ותיאור האירועים הוא 2 2 1 1 2 2 (כלומר, אבד עיפרון, נוסף עט, אבד עיפרון), אז פלט התוכנית צריך להיות הודעה המציינת כי בקלמר יש 5 עטים ו-8 עפרונות.

שאלה 11.5

כתבו מחלקה המגדירה את הטיפוס מעגל. תכונות של עצם מהמחלקה הן צבע המעגל ורדיוס המעגל. הפעולות שניתן לבצע על עצם מהמחלקה הן חישוב שטח, חישוב היקף והחזרת צבע המעגל.

כתבו תוכנית עזר לציור מעגלים. התוכנית תקלוט מהמשתמש את צבע המעגל ואת רדיוסו ותציג כפלט:

שטח המעגל הוא <שטח המעגל> היקף המעגל הוא <היקף המעגל> וצבעו הוא <צבע המעגל>

התוכנית תסתיים כאשר הצבע הנקלט יהיה שחור.

שאלה 11.6

פתחו וממשו מערכת לניהול מספרת כלבים. המערכת קולטת מספר שלם n ואחריו פרטים של n כלבים הבאים להסתפר במספרה. המערכת מציגה עבור כל כלב את סוג התספורת שיסופר בה. פרטים של כלב כוללים: שם כלב, אורך שיער (קצר או ארוך) וסוג נביחה ("הב", "הב-הב" או "הב-הב-הב"). סוגי התספורות האפשריים הם: גילוח, תספורת קצרה ותספורת רגילה.

המספרה מחליטה עבור כל כלב את סוג התספורת המתאימה לו: אם לכלב יש שיער קצר ונביחתו היא "הב" התספורת היא גילוח. אם לכלב יש שיער ארוך ונביחתו היא "הב הב" או "הב הב הב" הוא יסופר בתספורת קצרה. אחרת התספורת היא באורך רגיל.

הדרכה: התכונות של עצם מסוג כלב הם: שם, אורך שיער ונביחה. פעולות של עצם מסוג כלב הם: פעולה בונה ופעולות גישה המחזירות את ערכי התכונות. הפעולה הראשית תיצור עצמים מסוג כלב לפי נתונים המתקבלים מהקלט ותציג עבור כל כלב את סוג התספורת המתאימה לו.

בעיה 4

מטרת הבעיה ופתרונה: העמקה בעצמים

כתבו תוכנית המדמה משחק קובייה לזוג שחקנים. כל שחקן מטיל שתי קוביות בתורו. כל שחקן צובר את הנקודות מהטלות הקוביות שלו. "סיבוב" במשחק מורכב מתור של שחקן ראשון ואחריו תור של שחקן שני. המנצח הוא הראשון שמגיע ל-100 נקודות או יותר. אם שני השחקנים הגיעו ל-100 נקודות או יותר באותו "הסיבוב" נכריז על תיקו. אם שחקן מטיל דאבל (כגון 6-6) השחקן השני מקבל ניקוד כפול בתור הבא.

כתבו מחלקה המגדירה שחקן. לעצם מטיפוס שחקן נשמור את הניקוד המצטבר, ופעולותיו הן "שחק" ושתי פעולות גישה המאפשרות לברר את ניקוד השחקן ואם ניצח.

הגדרת המחלקה *שחקן*

נזדקק למחלקה המגדירה שחקן.

הגדרת התכונות

על פי הגדרת הבעיה, לעצם מהמחלקה שחקן תהיה תכונה השומרת את הניקוד המצטבר של השחקן. בהתאם לכך נבחר את המשתנה הבא לתיאור התכונה:

- **points** – משתנה מטיפוס שלם מייצג את הניקוד המצטבר שצבר השחקן עד כה.

הגדרת הפעולות

- **פעולה בונה** – הפעולה הבונה לא תקבל פרמטרים אלא רק תאתחל את הניקוד המצטבר.
- **Play** – הפעולה מקבלת פרמטר בוליאני הקובע אם לשחק כתור רגיל או כתור בניקוד כפול. הפעולה מדמה את זריקת שתי הקוביות, מעדכנת את הניקוד המצטבר של השחקן ומחזירה אם יצא דאבל.
- **GetPoints** – פעולת גישה המחזירה את הניקוד המצטבר של השחקן.
- **IsWin** – הפעולה איננה מקבלת פרמטרים והיא מחזירה **true** אם השחקן צבר 100 נקודות או יותר ו-**false** אחרת.

מימוש המחלקה

```
/*
המחלקה שחקן
*/
public class Player
{
    private int points;
    //פעולה בונה לשחקן//
    public Player()
    {
        points = 0;
    }
    // הפעולה מקבלת פרמטר בוליאני הקובע אם לשחק תור כרגיל או כתור
    // בניקוד כפול. הפעולה מדמה את זריקת הקוביות ומעדכנת את הניקוד
    // המצטבר של השחקן. הפעולה מחזירה אם יצא דאבל
    public bool Play(bool doublePoints)
    {
        Random rnd = new Random();
        int die1 = rnd.Next(1,7);
        int die2 = rnd.Next(1,7);
        int playPoints = die1 + die2;
        if (!doublePoints)
            points = points + die1 + die2;
        else
            points = points + 2 * (die1 + die2);
        return (die1 == die2);
    }
    //מחזירה את מספר הנקודות שצבר השחקן//
    public int GetPoints()
    {
        return points;
    }
    //מחזירה אמת אם הניקוד המצטבר גדול או שווה ל-100//
    public bool IsWin()
    {
        return (points >= 100);
    }
} // class Player
```

הגדרת הפעולה הראשית

כתבנו מחלקה המגדירה את הטיפוס שחקן, וכעת עלינו להמשיך לפיתוח הפעולה הראשית המדמה את המשחק.

פירוק לתת-משימות

את משימתה של הפעולה הראשית ניתן לפרק לתת-משימות באופן הבא:

1. יצירת השחקנים
2. דימוי המשחק

בחירת משתנים

player1 – משתנה מטיפוס שחקן, מייצג את שחקן 1.
player2 – משתנה מטיפוס שחקן, מייצג את שחקן 2.
doub – משתנה מטיפוס בוליאני מייצג האם השחקן זרק מספר כפול.

שימו ♥ לכך שלא ניתן לקרוא למשתנה double מפני שהמילה **double** היא מילה שמורה בשפה המציינת טיפוס ממשי. באופן כללי לא ניתן לקרוא למשתנים במילים שמורות בשפה.

כיצד נממש את התת-משימה השנייה?

האלגוריתם

1. כל עוד שחקן 1 לא ניצח וגם שחקן 2 לא ניצח בצע:
   1.1 אם לשחקן 2 יצא דאבל
      1.1.1 שחקן 1 – שחק כפול
   1.2 אחרת
      1.2.1 שחקן 1 - שחק
   1.3 אם לשחקן 1 יצא דאבל
      1.3.1 שחקן 2 – שחק כפול
   1.4 אחרת
      1.4.1 שחקן 2 – שחק
2. אם שחקן 1 ניצח וגם שחקן 2 ניצח
   2.1 הצג כפלט "תיקו"
3. אחרת אם שחקן 1 ניצח
   3.1 הצג כפלט "שחקן 1 ניצח"
4. אחרת
   4.1 הצג כפלט "שחקן 2 ניצח"

מימוש

```
/*
        המחלקה הראשית המממשת את המשחק
*/
using System;
public class DiceGame
{
    public static void Main()
    {
        Player player1 = new Player();
        Player player2 = new Player();
        bool doub = false;
        while (!player1.IsWin() && !player2.IsWin())
        {
             // שחקן ראשון משחק
             doub = player1.Play(doub);
             Console.WriteLine("player1 sum: {0}",
                                                    player1.GetPoints());
             // שחקן שני משחק
             doub = player2.Play(doub);
```

```
            Console.WriteLine("player2 sum: {0}",
                                                player2.GetPoints());
        }
        //הצגת הפלט
        if (player1.IsWin() && player2.IsWin())
            Console.WriteLine("tie");
        else if (player1.IsWin())
            Console.WriteLine("player 1 won");
        else
            Console.WriteLine("player 2 won");
    } // Main
} // class DiceGame
```

סוף פתרון בעיה 4

כפי שכבר ציינו, בעת הגדרת מחלקה ניתן לספק פעולות גישה המאפשרות לעדכן תכונות (Set) ולאחזר את ערכן (Get). לעתים נרצה לאפשר גישה חלקית בלבד או לא לאפשר גישה בכלל. למשל בפתרון בעיה 4, לתכונה points הוגדרה פעולת גישה המחזירה את ערכה, אך לא הוגדרה פעולת גישה המעדכנת את ערכה. הסיבה לכך היא, שאנו רוצים להגן על התכונה מפני עדכון מבחוץ. רק עצם מסוג שחקן יכול לעדכן את הניקוד שלו.

ההחלטה לאילו תכונות לאפשר גישה דרך פעולות גישה ואילו תכונות לא לחשוף כלל, מושפעת מהגדרת הבעיה ובפרט מתפקידן של התכונות במשימה שיש לפתור. הגדרה מבוקרת ומושכלת של פעולות גישה יכולה לסייע לנו בהגנה על הנתונים מפני שינוי בלתי מבוקר, וכן בהסתרת מידע פרטי מפני שאר חלקי התוכנית.

שאלה 11.7

במשחק "צבעי קלפים" מקבל כל משתתף 5 קלפים מכל צבע. הצבעים הם כחול, אדום וירוק. במשחק משתתפים 4 שחקנים. כל משתתף בתורו, זורק קלף בצבע כלשהו ולוקח קלף אחר מהקופה. המנצח הוא המשתתף הראשון שצבר 10 קלפים מאותו צבע.

פתחו וממשו תוכנית לניהול משחק "צבעי הקלפים". התוכנית תשתמש במחלקה "שחקן".

בכל תור התוכנית תפעיל על השחקן התורן את פעולת זריקת קלף, תגריל עבורו את צבע הקלף שיקבל מהקופה, ותעדכן את קלפיו בהתאם. התוכנית תסתיים כאשר אחד השחקנים ינצח.

הדרכה: לעצם מטיפוס המחלקה "שחקן" יש שלוש תכונות: עבור כל אחד משלושת הצבעים יש תכונה השומרת את מספר הקלפים שיש לשחקן מצבע זה. לעצם ממחלקה זו יש שתי פעולות: הוספת קלף וזריקת קלף. פעולת הוספת הקלף תקבל כפרמטר את צבע הקלף שיש להוסיף ותעדכן את התכונה המתאימה. פעולת זריקת קלף תחזיר את צבע הקלף שבחר השחקן לזרוק. הפעולה תבחר את הצבע שממנו יש לשחקן הכי פחות קלפים (אם יש יותר מצבע אחד כזה, יוחזר אחד מהם). יש כמובן צורך גם בפעולות גישה, כדי שהתוכנית תוכל לדעת את מצב הקלפים של השחקנים כדי שניתן יהיה לקבוע ניצחון. חשבו: מה צריכה לבצע הפעולה הבונה של עצם ממחלקה זו?

שאלה 11.8

במזללה "אוכל טעים" יש תפריט קבוע הכולל שתייה, מנה עיקרית ותוספת. השתייה יכולה להיות שתייה בכוס קטנה, בינונית או גדולה. המנה העיקרית יכולה להיות אחת מ-8 אפשרויות המשתנות כל יום, ולכן מסמנים אותה במספר בין 1 ל-8. התוספת היא צ'יפס, פירה או סלט. ארוחה רגילה כוללת שתייה בכוס בינונית, מנה עיקרית שמספרה 1 ופירה.

בצהרי כל יום נכנסים 20 סועדים למזללה. עובד הדלפק שואל את הסועד התורן באיזו ארוחה הוא מעוניין. אם הסועד אומר כי הוא מעוניין בארוחה רגילה הוא מקבל את הארוחה הרגילה. אחרת העובד שואל את הסועד איזה גודל של כוס שתייה, איזו מנה עיקרית ואיזו תוספת הוא רוצה להזמין. לפני ההגשה חוזר העובד על פרטי הארוחה המוזמנת, ומוסיף "בתיאבון".
פתחו וממשו תוכנית לניהול המזללה. הפעולה הראשית של התוכנית תשתמש במחלקה "ארוחה".

הדרכה: הגדירו מחלקת ארוחה הכוללת את התכונות: גודל שתייה, מנה עיקרית ותוספת. הפעולה הבונה של עצם מהמחלקה תקבל את פרטי הארוחה ותעדכן אותם. לצורך עמידה בדרישת החזרה על פרטי הארוחה המוזמנת לפני ההגשה יש להגדיר במחלקת הארוחה גם פעולה המחזירה כמחרוזת את תיאור הארוחה.

שאלה 11.9
בשקית סוכריות יש סוכריות מארבעה צבעים שונים (אדום, צהוב, ירוק וחום). קיימות שקיות בשני גדלים: שקית קטנה ושקית גדולה. בשקית קטנה יש 20 סוכריות ובשקית גדולה 28.
אם מספר הסוכריות מכל צבע בשקית שווה, תיקרא השקית **מאוזנת**. אם מספר הסוכריות מכל צבע בשקית שונה, תיקרא השקית **בלתי מאוזנת**.
פתחו וממשו אלגוריתם, שיקלוט את מספר השקיות המיוצרות במפעל ביום מסוים, ולאחר מכן יקלוט עבור כל שקית את תכולתה. תכולת השקית תיקלט כמחרוזת המורכבת מהתווים r (אדום), y (צהוב) g (ירוק) ו-b (חום) המייצגים את צבעי הסוכריות בשקית. האלגוריתם יציג כפלט את מספר השקיות הקטנות ואת מספר השקיות הגדולות שיוצרו במפעל באותו יום ואת מספר השקיות הבלתי מאוזנת מבין אלו שיוצרו באותו יום (ללא קשר לגודלן).

הדרכה: כתבו מחלקה המגדירה שקית סוכריות, ולמחלקה פעולה בונה המקבלת מחרוזת המכילה את כל הצבעים של הסוכריות בשקית. לעצם מטיפוס שקית סוכריות יהיו התכונות הבאות: תכונה אחת עבור כל צבע לשמירת מספר הסוכריות בצבע זה, ותכונה לשמירת גודל השקית (קטנה או גדולה). הפעולה הבונה תאתחל את ערכי התכונות. הגדירו את הפעולות הנדרשות לצורך קבלת המידע המבוקש.

11.3 תכונות מורכבות

בעיה 5

מטרת הבעיה ופתרונה: היכרות עם תכונות מורכבות של עצמים: מערך כתכונה.

במשחק ניחושים קיים טופס ובו רשומים 5 מספרים שונים בין 1 ל-100 וכן תיאור הפרס שזוכים בו אם מנחשים את כל המספרים הרשומים על הטופס. במשחק זה הפרס הוא מיליון דולר. פתחו וממשו תוכנית המתארת את משחק הניחושים. ראשית התוכנית תיצור טופס ניחושים. לאחר מכן התוכנית תקלוט מהשחקן מספר ותבדוק אם הוא קיים בטופס. אם המספר קיים ועדיין לא נוחש, התוכנית תציג כפלט את ההודעה "ניחוש נכון", אחרת תוצג ההודעה "ניחוש שגוי". התוכנית תסתיים לאחר שהשחקן ינחש נכונה את כל המספרים בטופס, או לאחר 10 ניחושים. אם השחקן ניחש נכונה את כל המספרים בטופס, התוכנית תציג כפלט את ההודעה "ניצחת: זכית ב<הפרס>", ואחרת, תוצג ההודעה "הפסדת: לא זכית ב<הפרס>".

לצורך פתרון הבעיה נכתוב מחלקה המגדירה טופס למשחק הניחושים.

הגדרת המחלקה *טופס ניחושים*

הגדרת התכונות

על-פי הגדרת הבעיה נגדיר לעצם מהמחלקה טופס ניחושים את התכונות: פרס ומספרים. התכונה מספרים מייצגת את רשימת המספרים בטופס שאותם המשתתף צריך לנחש.

כיצד נשמור את רשימת המספרים? נשמור מערך של מספרים שלמים שגודלו 5. אולם לא נוכל להסתפק ברשימת המספרים מכיוון שעבור כל מספר עלינו לדעת אם המשתתף כבר ניחש מספר זה או לא. לכן, המחלקה תכיל עוד תכונה. תכונה זו תהיה מערך בוליאני וגם הוא בגודל 5, שיחווה עבור כל מספר אם ניחשו אותו או לא. כלומר אם ניחשו את המספר השלישי אז האיבר השלישי במערך הבוליאני יהיה **true**.

שימו ♥: כאשר מגדירים מחלקה, התכונות יכולות להיות מכל טיפוס שהוא, גם מערך. הגדרת מערך כתכונה נעשית כהגדרת כל תכונה מטיפוס אחר.

נבחר את המשתנים הבאים לתיאור התכונות:

- **prize** – מטיפוס מחרוזת, מייצג את הפרס שזוכים בו אם מנחשים את כל המספרים בטופס.
- **numbers** – מערך של שלמים שונים בין 1 ל-100, מייצג את המספרים שהמשתתף צריך לנחש.
- **guessedNumbers** – מערך בוליאני, מייצג עבור כל מספר אם כבר ניחשו אותו או לא.
- **SIZE** – קבוע מטיפוס שלם, מייצג את מספר המספרים הנמצאים בטופס, במקרה זה 5.

הגדרת הפעולות

- **פעולה בונה** – הפעולה הבונה תקבל כפרמטר את הפרס ותאתחל אותו. מה לגבי אתחול המספרים? הפעולה הבונה תגריל את המספרים בטופס ותאתחל את מערך המספרים בהתאם. בהתחלה עדיין לא ניחשו אף מספר ולכן המערך הבוליאני guessedNumbers יאותחל כולו ב-**false**. שימו לב, עלינו לדאוג לכך שכל המספרים שנגריל יהיו שונים זה מזה. לצורך כך, נעזר במערך בוליאני שישמור עבור כל מספר מ-1 עד 100 אם כבר הגרילו אותו או לא. מערך זה הוא משתנה עזר לצורך הגרלת המספרים, לכן נגדיר אותו כמשתנה מקומי בפעולה הבונה ולא כתכונה של העצם.

  שימו ♥: פעולת ההגרלה נמשכת עד שמגרילים 5 מספרים שונים, אנו מניחים שפעולה זו תסתיים.

- **בדיקת ניחוש** – לאחר שהמשתתף מנחש מספר, הוא מקבל הודעה אם הוא צדק בניחושו או טעה. נגדיר פעולה IsGoodGuess המקבלת מספר כפרמטר. הפעולה תחזיר **true** אם המספר מופיע בטופס ועדיין לא נוחש ו-**false** אחרת. בנוסף לכך, אם המספר מופיע בטופס ועדיין לא נוחש, הפעולה תעדכן ל-**true** את התא המתאים במערך guessedNumbers.

- **בדיקת ניצחון** – כיצד נדע אם המשתתף ניחש את כל המספרים? נגדיר פעולה בוליאנית בשם IsWin. הפעולה תחזיר **true** אם המשתתף ניחש את כל המספרים בטופס, כלומר כל ערכי התאים במערך guessedNumbers הם **true**, אחרת יוחזר **false**.

- **פעולת גישה להחזרת הפרס** – על מנת להדפיס את הפרס בסוף המשחק נגדיר פעולת גישה GetPrize, להחזרת הפרס. פעולה זו אינה מקבלת ערכים והיא מחזירה את הפרס.

מימוש המחלקה

```
/*
המחלקה טופס-למשחק-הניחושים
*/
public class GuessForm
{
    private string prize;     //שומר את הפרס
    private int[] numbers;    //מערך המספרים שאותם צריך לנחש
    private bool[] guessedNumbers;
    //מערך המחווה לגבי כל מספר האם ניחשו אותו כבר או לא
    private const int SIZE = 5;  //קבוע לציון מספר המספרים שבטופס
    //  פעולה בונה - מקבלת את הפרס, מגרילה את המספרים בטופס
    //  ומאתחלת את המערך המחווה אם המספרים נוחשו
    public GuessForm(string prize)
    {
        this.prize = prize;
        numbers = new int[SIZE];
        guessedNumbers = new bool[SIZE];
        Random rnd = new Random();
        //מערך עזר בוליאני המשמש לבחירת מספרים שונים
        bool[] nums = new bool[101];
        for (int i = 0; i <= 100; i++)
            nums[i] = false;
        int r;
        for (int i = 0; i < SIZE; i++)
        {
            r = rnd.Next(1,101);
            while (nums[r] == true)
                 r = rnd.Next(1,101);
            nums[r] = true;
            numbers[i] = r;
            guessedNumbers[i] = false;
        }
    }
    //פעולת גישה המחזירה את הפרס
    public string GetPrize()
    {
        return prize;
    }
    //פעולה הבודקת האם כל המספרים בטופס נוחשו
    //מחזירה אמת אם כן
    public bool IsWin()
    {
        for (int i = 0; i < SIZE; i++)
             if (guessedNumbers[i] == false)
                  return false;
        return true;
    }
    //פעולה המקבלת כפרמטר מספר ובודקת האם הוא נמצא בין המספרים בטופס,
    //מעדכנת את מערך המספרים שכבר נוחשו ומחזירה אמת אם המספר נמצא
    //בטופס ועדיין לא נוחש, אחרת שקר
```

```
    public bool IsGoodGuess(int guess)
    {
        for (int i = 0; i < SIZE; i++)
        {
            if (numbers[i] == guess)
            {
                if (! guessedNumbers[i])
                {
                     guessedNumbers[i] = true;
                     return true;
                }
                else
                     return false;
            }
        }
        return false;
    }
}//class GuessForm
```

שימו ♥: בפעולה הבונה אנו מצהירים על משתנים כמו rnd וכמו nums המשמשים להגרלת המספר ולבדיקת כפילויות. משתנים אלה מוצהרים בתוך התחום של הפעולה הבונה. משתנים כאלה נקראים **משתנים מקומיים** (לוקאליים), משום שהם קיימים רק בתוך הפעולה. פעולות אחרות, אפילו של אותה מחלקה, לא יכולות להתייחס אליהם.

כאשר הפעולה מסתיימת המשתנים המקומיים שלה אינם קיימים יותר. כאשר תופעל הפעולה שוב יוקצה שוב שטח זיכרון עבור כל משתנה מקומי (שאינו בהכרח אותו שטח שהוקצה בהפעלה קודמת).

הדבר דומה למשתנה הבקרה שאנו מגדירים בתוך לולאת **for**. למשל בהוראה:

```
for (int i = 0; i < SIZE; i++)
```

מוצהר משתנה i המשמש בתוך הלולאה, אך מפסיק להתקיים עם סיומה.

למעשה בתוך כל תחום המוגדר בסוגריים מסולסלים (כמו למשל גוף של לולאה או גוף של הוראה לביצוע-בתנאי) ניתן להצהיר על משתנים מקומיים לאותו תחום. משתנים אלה מפסיקים להתקיים עם סיום ביצוע התחום.

הגדרת הפעולה הראשית

פירוק לתת-משימות

המשימה של הפעולה הראשית היא לממש את משחק הניחושים. את משימתה של הפעולה הראשית ניתן לפרק לתת-משימות, באופן הבא:

1. יצירת טופס ניחושים ואתחולו בפרס המתאים
2. המשחק עצמו
3. הצגת הפלט

בחירת משתנים

prize– משתנה מטיפוס מחרוזת, מייצג את הפרס.

guessForm – עצם מטיפוס "טופס ניחושים", מייצג את הטופס למשחק

counter – משתנה מטיפוס שלם, שומר את מספר הניחושים עד כה

guess – משתנה מטיפוס שלם, לתוכו נקלט הניחוש הנוכחי

האלגוריתם

1. צור עצם מטיפוס "טופס ניחושים" ואתחל את הפרס
2. כל עוד לא נוחשו כל המספרים וגם מונה הניחושים קטן מ-10 בצע:
   - 2.1 הגדל את מונה הניחושים ב-1
   - 2.2 קלוט מהמשתמש את הניחוש והשם ב-**guess**
   - 2.3 אם הניחוש מופיע בטופס
     - 2.3.1 הצג כפלט "ניחוש נכון"
   - 2.4 אחרת
     - 2.4.1 הצג כפלט "ניחוש שגוי"
3. אם כל המספרים נוחשו
   - 3.1 הצג כפלט: ניצחת: זכית ב<הפרס>
4. אחרת
   - 4.1 הצג כפלט: הפסדת: לא זכית ב<הפרס>

מימוש

```
/*
     המחלקה הראשית המממשת את משחק הניחושים
*/
using System;
public class GuessGame
{
    public static void Main()
    {
         string prize = "1 million dollars";
         GuessForm guessForm = new GuessForm(prize);
         int guess;
         int i = 0;
         while ((i < 10) && (guessForm.IsWin() == false))
         {
              i++;
              Console.Write("Please enter your guess: ");
              guess = int.Parse(Console.ReadLine());
              if (guessForm.IsGoodGuess(guess))
                   Console.WriteLine("Good Guess");
              else
                   Console.WriteLine("Bad Guess");
         }
         if (guessForm.IsWin())
              Console.WriteLine("You win: {0}",guessForm.GetPrize());
         else
              Console.WriteLine("You lose:{0}",guessForm.GetPrize());
    }// Main
}// GuessGame
```

סוף פתרון בעיה 5

שאלה 11.10

בתחרות קפיצה לגובה כל משתתף קופץ 5 קפיצות. כתבו מחלקה המגדירה קופץ לגובה. עצם מהמחלקה ישמור את שמו של הקופץ ואת תוצאות הקפיצות שלו.
פתחו וממשו תוכנית לאימון בקפיצה לגובה. התוכנית תקלוט את שמו של הקופץ. לאחר מכן תבקש התוכנית מהקופץ להכניס בכל פעם את תוצאת קפיצתו. (שימו לב שתוצאה של קפיצה לגובה היא מספר ממשי, למשל 1.15 מטר).
התוכנית תציג כפלט את שמו של הקופץ יחד עם תוצאת הקפיצה הטובה ביותר שלו.

שאלה 11.11

עזרו למורה נחמה להציג את הממוצע הכיתתי במקצועות חשבון ועברית. כתבו מחלקה המגדירה את ציוני הכיתה. בכיתה 20 תלמידים. עבור כל תלמיד המורה תכניס את ציונו בחשבון ואחר כך בעברית.

הדרכה: הגדירו במחלקה את הפעולות הבאות: פעולה לעדכון נתוני תלמיד המקבלת מספר סידורי של תלמיד ואת ציוניו בשני המקצועות, ופעולות המחזירות את הממוצע הכיתתי בכל מקצוע.

שאלה 11.12

במסעדת "יאמיאמי" מגישים: סלט, ספגטי, שניצל, המבורגר, מרק ועוגה. בכל יום בשעה 13:00 מגיעים סועדים רבים למסעדה. הטבח מוכן להתחיל לבשל רק לאחר שכל ההזמנות נעשו. כתבו מחלקה המגדירה את ניהול ההזמנות במסעדת "יאמיאמי". עצם מהמחלקה יכיל את רשימת המאכלים שמגישים במסעדה ובעבור כל מאכל כמה סועדים הזמינו אותו. פתחו וממשו תוכנית שתקלוט מכל סועד את הזמנתו (מספר בין 1 ל-6 המציין בהתאמה את המנות האפשריות) ותציג כפלט את כל המנות שצריך הטבח להכין וכמה מכל מנה. סוף הקלט יצוין בהזמנת מנה שמספרה הוא 0.

בעיה 6

מטרת הבעיה ופתרונה: היכרות עם מבני נתונים בסיסיים של עצמים (מערך).

כתבו תוכנית לניהול תחרות קפיצה לגובה. התוכנית תקלוט את מספר המשתתפים בתחרות. לאחר מכן ייקלטו נתוני המשתתפים: עבור כל קופץ לגובה תקלוט התוכנית את שמו של הקופץ, את מספר תעודת הזהות שלו ואת גובה קפיצתו.
התוכנית תציג כפלט את שמותיהם של כל זוגות המתחרים שגובה קפיצתם זהה, ואת גובה הקפיצה.
למשל, אם בתחרות יש 4 משתתפים, ולהם הנתונים הבאים:

שם	ת"ז	גובה קפיצה
פלוטו	1111	1.25
מיני	1112	2.02
דונלד	1120	1.25
מיקי	1121	1.25

אז בפלט צריכים להיות מוצגים הזוגות הבאים:
- פלוטו ודונלד קפצו לגובה 1.25
- פלוטו ומיקי קפצו לגובה 1.25

הגדרת המחלקה *קופץ לגובה*

הגדרת התכונות

על פי הגדרת הבעיה לעצם מהמחלקה קופץ לגובה יש התכונות הבאות: שם מלא, ת"ז וגובה קפיצה.

כיצד נשמור את מספר תעודת הזהות של הקופץ? מכיוון שמספר תעודת הזהות משמש כמזהה, כלומר הוא איננו מספר שאמורים לבצע עליו פעולות חשבוניות, נשמור את מספר תעודת הזהות כמחרוזת.

נבחר את המשתנים הבאים לתיאור התכונות:

- **name** – מחרוזת המייצגת את שם הקופץ.
- **id** – מחרוזת המייצגת את מספר תעודת הזהות של הקופץ.
- **jumpHeight** – משתנה מטיפוס ממשי המייצג את גובה הקפיצה של הקופץ.

הגדרת הפעולות

- **פעולה בונה** – הפעולה הבונה תקבל כפרמטרים את שם הקופץ ואת מספר תעודת הזהות שלו ותאתחל את התכונות המתאימות. מה לגבי אתחול גובה הקפיצה של הקופץ? בהתחלה הקופץ עדיין לא קפץ. לכן גובה הקפיצה יאותחל ב-0.
- **עדכון גובה קפיצה** – לאחר שהקופץ יקפוץ נרצה לעדכן את גובה הקפיצה שלו. לכן נרצה להגדיר פעולת גישה SetJumpHeight המעדכנת את גובה הקפיצה.
- **קבלת גובה קפיצה** – לצורך מציאת כל זוגות הקופצים שגובה קפיצתם זהה, נגדיר פעולת גישה GetJumpHeight, המחזירה את גובה הקפיצה של הקופץ.
- **קבלת שם קופץ** – על מנת להציג את שמותיהם של הקופצים שגובה קפיצתם זהה נגדיר פעולת גישה GetName המחזירה את שמו של הקופץ.

מימוש המחלקה

```
/*
מחלקת קופץ לגובה
*/
public class Jumper
{
      // תכונות הקופץ
      private string name;            // שם
      private string id;              // ת"ז
      private double jumpHeight;      // גובה קפיצה
      // הפעולה הבונה
      public Jumper(string name, string id)
      {
            this.name = name;
            this.id = id;
            jumpHeight = 0;
      }
      // פעולת גישה: מחזירה את שם הקופץ
      public string GetName()
```

```
    {
        return name;
    }
    //פעולת גישה: מחזירה את גובה הקפיצה של הקופץ
    public double GetJumpHeight()
    {
        return jumpHeight;
    }
    //פעולת גישה: מעדכנת את גובה הקפיצה של הקופץ
    public void SetJumpHeight(double jumpHeight)
    {
        this.jumpHeight = jumpHeight;
    }
} // class Jumper
```

הגדרת הפעולה הראשית

פירוק לתת-משימות

המשימה של הפעולה הראשית היא לקלוט את מספר הקופצים בתחרות. לאחר מכן, לקלוט את הנתונים של הקופצים ולהציג את שמותיהם של כל זוגות המתחרים שגובה קפיצתם זהה ואת גובה הקפיצה. את משימתה של הפעולה הראשית ניתן לפרק לתת-משימות, באופן הבא:

1. קליטת מספר הקופצים
2. קליטת הנתונים של הקופצים
3. הצגת כל זוגות המתחרים שגובה קפיצתם זהה והצגת גובה הקפיצה.

בחירת משתנים

את הקופצים נשמור במערך שבו כל איבר יהיה מטיפוס "קופץ לגובה". אלה המשתנים הדרושים:

jumpers – מערך מטיפוס "קופץ לגובה", מייצג את הקופצים לגובה אשר משתתפים בתחרות.

jumpersNum – משתנה מטיפוס שלם, שומר את מספר הקופצים.

jumpHeight – משתנה מטיפוס ממשי, שומר את גובה הקפיצה.

jumperName – מחרוזת, שומרת את שם הקופץ.

jumperId – מחרוזת, שומרת את מספר תעודת הזהות של הקופץ.

האלגוריתם

משימת קליטת נתוני הקופצים דומה למשימות קלט מבעיות קודמות. ההבדל היחיד הוא שבמקרה זה הקופצים הם עצמים ולא משתנים פשוטים. לכן עבור כל קופץ ניצור עצם מטיפוס "קופץ לגובה" ונאתחל אותו בשם ובמספר תעודת הזהות שהתקבלו עבורו כקלט. העצם שנוצר יישמר במערך הקופצים. לאחר מכן נקלוט את גובה הקפיצה עבור הקופץ ונעדכן אותו.

? כיצד נבצע את התת-משימה השלישית? כיצד נמצא את כל זוגות המתחרים אשר גובה קפיצתם זהה?

נעבור על המערך, ועבור כל קופץ נשווה את גובה קפיצתו לגובה קפיצתם של הקופצים האחרים. כלומר נעבור על המערך בלולאה מקוננת. הלולאה החיצונית תגדיר קופץ ועבור כל קופץ נבצע לולאה פנימית, שעוברת על רשימת הקופצים ומחפשת קופצים אחרים שגובה קפיצתם זהה לזה שלו. נרצה להשוות כל זוג קופצים פעם אחת בלבד. אחרי שהשווינו את גובה הקפיצה של הקופץ

הראשון לזה של הקופץ השני, לא נרצה להשוות אחר כך את גובה הקפיצה של הקופץ השני לזה של הראשון. לכן הלולאה הפנימית לא תתחיל בכל פעם מהקופץ הראשון, אלא מהקופץ שבא אחרי הקופץ התורן בלולאה החיצונית.

האלגוריתם ליישום התת-משימה השנייה הוא:

1. עבור כל i מ-0 עד מספר הקופצים פחות 2 בצע:
   1.1. עבור כל j מ-i+1 עד מספר הקופצים פחות 1 בצע:
      1.1.1. אם גובה הקפיצה של הקופץ ה-i במערך שווה לגובה הקפיצה של הקופץ ה-j במערך
         1.1.1.1. הצג את שם הקופץ ה-i והקופץ ה-j במערך ואת גובה קפיצתם.

מימוש

```
/*
    מחלקה ראשית הממששת תחרות קפיצה לגובה
*/
using System;
public class JumperContest
{
    public static void Main()
    {
        // הגדרה והקצאת משתנים
        int jumpersNum;
        Jumper[] jumpers;
        double jumpHeight;
        string jumperName;
        string jumperId;
        Console.Write("Enter number of jumpers: ");
        jumpersNum = int.Parse(Console.ReadLine());
        jumpers = new Jumper[jumpersNum];
        for (int i = 0; i < jumpersNum; i++)
        {
            Console.Write("Enter Jumper Name: ");
            jumperName = Console.ReadLine();
            Console.Write("Enter Jumper Id: ");
            jumperId = Console.ReadLine();
            jumpers[i]= new Jumper(jumperName, jumperId);
            //יצירת כל איבר במערך בתורו
            Console.Write("Enter Jump Height: ");
            jumpHeight = int.Parse(Console.ReadLine());
            jumpers[i].SetJumpHeight(jumpHeight);
        }
        // פלט
        for (int i = 0; i < jumpersNum-1; i++)
        {
            for (int j = i+1; j < jumpersNum; j++)
            {
                if (jumpers[i].GetJumpHeight() ==
                        jumpers[j].GetJumpHeight())
                {
```

```
                    Console.WriteLine("{0} and {1} jumped the same
                        height: {2}", jumpers[i].GetName(),
                        jumpers[j].GetName(),
                        jumpers[i].GetJumpHeight());
                } // if
            } // for j
        } // for i
    } // Main
} // class JumperContest
```

סוף פתרון בעיה 6

בפתרון בעיה 6 ראינו שניתן להגדיר מערך מכל טיפוס שהוא, בפרט מערך של עצמים ממחלקות שהגדרנו אנחנו. הגדרת מערך מטיפוס מחלקה מסוימת נעשה כהגדרת כל מערך מטיפוס אחר. עם זאת מערך של עצמים הוא למעשה מערך של הפניות לעצמים, ולכך יש חשיבות רבה, כפי שנדגים כעת. נתבונן בקטע התוכנית הבא:

```
Jumper[] jumpers = new Jumper[10];
Jumper j = new Jumper("Yoyo","99999");
jumpers[0] = j;
```

כתוצאה מביצוע קטע זה נוצר עצם אחד מסוג קופץ, אך לעצם זה יש שתי הפניות – המשתנה j ו-jumpers[0] – שניהם מפנים לאותו הקופץ ששמו Yoyo.
אם נעדכן את גובה הקפיצה באופן הבא:

```
j.SetJumpHeight(1.85);
```

נוכל לראות את השינוי גם דרך jumpers[0] כי הוא מפנה לאותו העצם.
בעקבות הוראת הפלט הבאה:

```
Console.WriteLine("jump height = {0}", jumpers[0].GetJumpHeight());
```

יוצג הפלט:

```
jump height = 1.85
```

מדוע? הרי לכאורה לא ביצענו שינוי באיברי המערך jumpers! הסיבה היא שהמערך jumpers הוא מערך של עצמים. כפי שאמרנו; כאשר איברי המערך הם מטיפוס מחלקה כלשהי, המערך הוא מערך של **הפניות** לעצמים. כלומר כל איבר במערך הוא הפניה לעצם. בקטע התוכנית שלעיל התבצע שינוי בעצם ש-j מפנה אליו. מכיוון שהתא הראשון במערך מפנה לאותו העצם, כל שינוי שנעשה באמצעות j ייראה גם מתוך המערך, ולחילופין כל שינוי שנעשה באמצעות jumpers[0] ייראה דרך j.

אם נרצה למנוע יצירת שתי הפניות לאותו העצם, נוכל ליצור את העצמים כפי שעשינו בפתרון הבעיה שהוצג לעיל:

```
jumpers[i] = new Jumper("Yoyo","99999");
```

בצורה זו ההפניה היחידה לקופץ נמצאת במערך.

שאלה 11.13

פתחו וממשו תוכנית למשחק בול פגיעה. התוכנית תגריל מספר בן 4 ספרות (ללא חזרות) והמשתמש ינסה לנחש את המספר. עבור כל ניחוש התוכנית תציג כמה ספרות הן "בול" וכמה ספרות הן רק "פגיעה".

נגדיר "בול" כשהמשתמש ניחש את הספרה במקומה הנכון ו"פגיעה" כשניחש ספרה המופיעה במספר שהוגרל אך לא במיקומו הנכון.

לדוגמא: המספר שבחרה התוכנית הוא 3456

המספר שניחש המשתמש הוא 2465

לכן הפלט יהיה: בול אחד (הספרה 4) ו- 2 פגיעות (הספרות 5 ו 6)

התוכנית תמשיך לקלוט מהמשתמש מספרים עד אשר ינחש את המספר, ורק אז תסתיים.

הדרכה: כתבו מחלקה המגדירה את המספר הסודי שרוצים לנחש. הפעולה הבונה תגריל מספר בן 4 ספרות ותשמור אותו במערך של שלמים בגודל 4 – כל תא במערך ייצג ספרה במספר. הגדירו במחלקה, פעולה המקבלת כפרמטר מספר בן 4 ספרות, בודקת כמה מהספרות הן בול וכמה קליעה ומחזירה מחרוזת עם התוצאה. במקרה שהניחוש נכון תוחזר המחרוזת "finished".

שאלה 11.14

בתחרות ניווטים הוחלט שהמשתתפים שיעלו לשלב הגמר הם אלה שהגיעו ליעד בזמן הנמוך מזמן ההגעה הממוצע של כל המשתתפים.

עליכם לפתח אלגוריתם שיקלוט את מספר המשתתפים, ולכל משתתף את הנתונים הבאים: שם, כתובת, מספר תעודת זהות וזמן ההגעה ליעד. האלגוריתם יציג כפלט את פרטי המשתתפים שיעלו לשלב הגמר, את כתובותיהם ואת מספרי תעודות הזהות שלהם.

א. הגדירו את המחלקה המתאימה למשתתף בתחרות, את כל התכונות ואת פעולות הגישה הנדרשות.

ב. פתחו אלגוריתם שהקלט שלו הוא זמן ההגעה ליעד של כל משתתף והפלט שלו הוא רשימת המשתתפים שזמן ההגעה שלהם ליעד נמוך מזמן ההגעה הממוצע.

ג. כתבו תוכנית הקולטת את מספר המשתתפים בתחרות. ממשו את האלגוריתם שפיתחתם בסעיף ב, בשימוש במחלקה שהגדרתם בסעיף א.

שאלה 11.15

כתבו תוכנית למציאת הילדים שלהם יש שמות ייחודיים בכיתה. התוכנית תקלוט את מספר הילדים בכיתה. לאחר מכן, תקלוט התוכנית עבור כל ילד את שמו הפרטי, את המקצוע האהוב עליו ואת גילו. התוכנית תציג את הנתונים של הילדים אשר שמותיהם ייחודיים בכיתה. ילד הוא בעל שם ייחודי אם בכיתה אין עוד ילד בשם זהה.

שאלה 11.16

א. נתונה התוכנית הבאה:

```
using System;
public class Animal
{
    private string kind = ""; // סוג
    public void SetKind(string Kind)
    {
        this.kind = Kind;
    }
    public string GetKind()
    {
        return kind;
    }
} // class Animal
```

```
public class MyAnimals
{
    public static void Main()
    {
         Animal ani = new Animal();
         Animal [] myAnimals = new Animal[3];
         ani.SetKind("dog");
         myAnimals[0] = ani;
         ani.SetKind("cat");
         myAnimals[1] = ani;
         ani.SetKind("fish");
         myAnimals[2] = ani;
         Console.WriteLine("My animals are {0} {1} {2}",
              myAnimals[0].GetKind(), myAnimals[1].GetKind(),
              myAnimals[2].GetKind());
    } // Main
} // class MyAnimals
```

מה הפלט שיוצג בסוף התוכנית? השלימו:

```
My animals are _________   _________   ________
```

ב. תקנו את הפעולה הראשית (אך לא את הוראת הפלט) כך שבסוף התוכנית תוצג ההודעה:

```
My animals are dog cat fish
```

שאלה 11.17

נתון קטע התוכנית הבא:

```
int[] intArray = new int[1];
int i = 10;
intArray[0] = i;
Console.WriteLine("intArray[0] is {0}", intArray[0]);
i = 5;
Console.WriteLine("intArray[0] is {0}", intArray[0]);
```

כתבו מהו הפלט של קטע התוכנית.

בעיה 7

מטרת הבעיה ופתרונה: היכרות עם מחלקת שירות ועם פעולות סטטיות.

פתחו וממשו מחשבון המבצע פעולות מתמטיות מתקדמות הכוללות: העלאת מספר בחזקה, בדיקה אם מספר הוא חזקה של מספר אחר וחישוב סכום הספרות של מספר.
פתחו תוכנית לשימוש במחשבון. התוכנית תציג לפני המשתמש את הפעולות שהמחשבון יכול לבצע. המשתמש יבחר את הפעולה הרצויה ויספק את הקלט המתאים לפעולה. בסיום ביצוע הפעולה תציג התוכנית פלט הכולל את פרטי התרגיל שבוצע ואת התוצאה. למשל עבור העלאת 2 בחזקת 5 יוצג הפלט: 2^5=32.

הגדרת המחלקה מחשבון

על פי הגדרת הבעיה נראה שלעצם מסוג מחשבון אין תכונות כלשהן. מחשבון יכול לבצע פעולות חשבוניות על מספרים הניתנים לו כפרמטרים. עד כה ראינו מחלקות המורכבות מתכונות ומפעולות. אולם יש מקרים שמחלקה תכיל רק פעולות. במקרים כאלו, נקרא למחלקה "מחלקת

שירות", מפני שהיא מכילה אוסף של שירותים (פעולות) שאינם פועלים על עצם כלשהו. פעולות אלה יכולות לקבל פרמטרים ולהחזיר ערך אך הן לא פועלות על תכונות. פעולות אלה נקראות **פעולות מחלקה** (class methods) או **פעולות סטטיות** (static methods).

המחלקה Math שהשתמשנו בה בעבר, היא מחלקת שירות הכוללת פעולות סטטיות המאפשרות לבצע חישובים מתמטיים שונים. כדי להשתמש בפעולות סטטיות איננו צריכים ליצור עצם מהמחלקה אלא להשתמש במחלקה עצמה ולהפעיל עליה ישירות את הפעולות, למשל:

```
int x = Math.Sqrt(4);
```

לעומת זאת כאשר רוצים להפעיל פעולה של עצם, כלומר פעולה שאיננה סטטית, אנו יוצרים עצם מהמחלקה (באמצעות ההוראה **new**) ועליו אנו מפעילים את הפעולה, למשל:

```
Random rnd = new Random();
int val = rnd.Next(6);
```

במחלקת שירות כל הפעולות מוגדרות כפעולות סטטיות. כלומר כעת כותרות הפעולות ייראו כך:

```
public static טיפוס-הערך-המוחזר שם-הפעולה(רשימת פרמטרים)
```

כעת, ניצור מחלקת שירות בשם מחשבון.

הגדרת הפעולות

- **העלאת מספר בחזקה**: הפעולה תקבל שני פרמטרים מטיפוס שלם, המייצגים בסיס ומעריך חיובי. הפעולה תחשב ותחזיר את הבסיס בחזקת המעריך. החישוב יתבצע באמצעות פעולת הכפל ולא באמצעות הפעולה Pow של המחלקה Math.
- **האם מספר הוא חזקה של מספר אחר**: הפעולה תקבל שני פרמטרים מטיפוס שלם ותבדוק אם המספר הראשון הוא חזקה של המספר השני. אם כן הפעולה תחזיר **true** אחרת יוחזר **false**. למשל אם התקבלו המספרים 25 ו-5 יוחזר הערך **true** כי 25 הוא חזקה של 5.
- **חישוב סכום הספרות של מספר**: הפעולה תקבל מספר שלם ותחזיר את סכום ספרותיו.

שימו ♥: מכיוון שזוהי מחלקת שירות אין צורך בפעולה בונה.

מימוש המחלקה

```
/*
מחלקת שירות: מחשבון
*/
public class Calculator
{
    //פעולה המקבלת בסיס ומעריך ומחזירה את הבסיס בחזקת המעריך
    public static int Power(int b, int exponent)
    {
        if (exponent == 0)
            return 1;
        int result = b;
        for (int i = 1; i < exponent; i++)
            result = result * b;
        return result;
    }
    // :פעולה המקבלת שני מספרים שלמים ומחזירה
    // האם הראשון הוא חזקה של השני
    public static bool IsPower(int result, int b)
    {
```

```
        double div = result;
        if (result == 1)
             return true;
        if (result % b != 0)
             return false;
        while(result > 1)
        {
             div = div / b;
             result = result / b;
        }
        return (div == 1);
    }
    //פעולה המקבלת מספר שלם ומחזירה את סכום ספרותיו
    public static int DigitSum(int num)
    {
        int sum = 0;
        while (num != 0)
        {
             sum = sum + num % 10;
             num = num / 10;
        }
        return sum;
    }
}// Calculator
```

שימו ♥: לצורך ביצוע הפעולה DigitSum השתמשנו בתבנית "פירוק מספר לספרותיו".

הגדרת הפעולה הראשית

המשימה של הפעולה הראשית היא להציג את הפעולות שניתן לבצע במחשבון, לקלוט את הפעולה שמבקש המשתמש לבצע, לקלוט מהמשתמש את הפרמטרים הנחוצים לביצוע הפעולה ולהציג את הפרמטרים שנקלטו ואת תוצאת הפעולה.

בחירת משתנים

operation – משתנה מטיפוס שלם, מייצג את הפעולה החשבונית שהמשתמש רוצה לבצע.

num – משתנה מטיפוס שלם, מייצג את הפרמטר הראשון לפעולה שנבחרה.

num1 – משתנה מטיפוס שלם, מייצג (במקרה הצורך) את הפרמטר השני לפעולה שנבחרה.

מימוש

```
/*
        מחלקה ראשית המשתמשת במחלקת המחשבון
*/
using System;
public class CalculatorTest
{
    public static void Main()
    {
        int operation;
        int num;
        int num1;
```

```
        Console.WriteLine("Please enter the number of the "+
                          "operation you would like to perform:");
        Console.WriteLine("1: power");
        Console.WriteLine("2: is power");
        Console.WriteLine("3: digit sum");
        operation = int.Parse(Console.ReadLine());
        switch (operation)
        {
             case 1:
                  Console.Write("Please enter the base number: ");
                  num = int.Parse(Console.ReadLine());
                  Console.Write("Please enter the exponent: ");
                  num1 = int.Parse(Console.ReadLine());
                  Console.WriteLine("{0}^{1}={2}", num, num1,
                                       Calculator.Power(num,num1));
             break;
             case 2:
                  Console.Write("Please enter the result: ");
                  num = int.Parse(Console.ReadLine());
                  Console.Write("Please enter the base: ");
                  num1 = int.Parse(Console.ReadLine());
                  if (Calculator.IsPower(num,num1))
                       Console.WriteLine("{0} is a power of {1}", num
                                                            , num1);
                  else
                       Console.WriteLine("{0} is not a power of {1}",
                                                         num, num1);
             break;
             case 3:
                  Console.Write("Please enter a number: ");
                  num = int.Parse(Console.ReadLine());
                  Console.WriteLine("The digit sum of {0} is {1}",
                                    num, Calculator.DigitSum(num));
             break;
             default:
                  Console.WriteLine("Wrong option");
             break;
        }// Switch
    }// Main
}// ClaculatorTest
```

סוף פתרון בעיה 7

שימו ♥: הפעולות במחלקת המחשבון מוגדרות כפעולות סטטיות, לכן השתמשנו בהן **ללא** יצירת עצם מסוג מחשבון. זימון פעולות סטטיות נעשה ישירות דרך שם המחלקה:

```
שם המחלקה.שם הפעולה(פרמטרים)
```

למשל:

```
Calculator.DigitSum(num)
```

נסתכל על מימוש הפעולה DigitSum במחלקה "מחשבון". הפעולה מקבלת פרמטר בשם num שהוא המספר וסוכמת את ספרותיו. לצורך ביצוע הפעולה, חילקנו שוב ושוב את num ב-10 עד שערכו היה שווה לאפס.

בפעולה הראשית, קולטים ערך למשתנה num ומעבירים אותו כפרמטר לפעולה DigitSum. לאחר סיום הפעולה המשתנה num מוצג בפלט וניתן לראות שערכו לא השתנה. מדוע? כאשר אנו מעבירים משתנה מטיפוס פשוט כפרמטר, ערכו של המשתנה הוא זה שמועבר ולא המשתנה עצמו. כלומר, כל שינוי בתוך הפעולה על הפרמטר משנה את הפרמטר בלבד אך לא את המשתנה שהועבר. לצורה זו של העברת פרמטרים קוראים **call by value** כי **הערך** של המשתנה הוא זה שמועבר ולא המשתנה עצמו. נרחיב בנושא זה בבעיה 9 בפרק זה.

שאלה 11.18

א. נתונות המחלקות הבאות:

```
using System;
public class Doubler
{
    public static void DoubleIt(int num)
    {
         num = num * 2;
         Console.WriteLine("Doubled: num = {0}", num);
    }
}// Doubler

public class DoublerTest
{
    public static void Main()
    {
         int num = 2;
         Console.WriteLine("num is {0}", num);
         Doubler.DoubleIt(num);
         Console.WriteLine("num after double is {0}", num);
    } // Main
}//DoublerTest
```

מה הפלט שיוצג כתוצאה מביצוע הוראת הפלט האחרונה בתוכנית? השלימו:

```
num after double is _________
```

זכרו שפרמטרים ב-#C מועברים על פי ערך (call by value) ולכן ערכו של `num` לא ישתנה כתוצאה מביצוע הפעולה.

ב. הדרך היחידה שבה הפעולה `DoubleIt` יכולה להכפיל את ערכו של `num` היא בהחזרת הערך `num*2` והשמת הערך המוחזר במשתנה `num`. שנו את הפעולה `DoubleIt` (ואת אופן הפעלתה), כך שהפעולה הראשית תקבל את ערכו של num מוכפל ב-2. כלומר, כתוצאה מביצוע הוראת הפלט האחרונה בתוכנית תוצג ההודעה:

```
num after double is 4
```

שאלה 11.19

פתחו וממשו מחלקת שירות למחרוזות. פעולות המחלקה הן:

1. האם פלינדרום – פעולה המקבלת מחרוזת ומחזירה אם היא פלינדרום. מחרוזת היא פלינדרום כאשר רצף התווים משמאל לימין זהה לרצף התווים מימין לשמאל. למשל המחרוזות "aba" ו-"xyyx" הן פלינדרום ואילו המחרוזת "abab" אינה פלינדרום.
2. האות השכיחה ביותר – פעולה המקבלת מחרוזת ומחזירה את האות השכיחה ביותר במחרוזת. אם יש כמה כאלה תוחזר הראשונה מביניהן.

3. הפיכת המחרוזת – פעולה המקבלת מחרוזת ומחזירה את המחרוזת ההפוכה.

בעיה 8

מטרת הבעיה ופתרונה: העברת עצם כפרמטר לפעולה.

החייזר בונבון הלך לאיבוד בגלקסיה, מלך הגלקסיה מצא אותו והוא מוכן להחזירו רק למי שבונבון יזהה כקרוב משפחתו. רבים באו בטענות שהם קרובי משפחתו של בונבון.

חייזר קובע שחייזר אחר הוא קרוב משפחה רק אם לשניהם שם משפחה זהה, שניהם הגיעו מאותו כוכב ולשניהם מאכל אהוב זהה.

שם המשפחה של בונבון הוא אלף, הוא הגיע מכוכב מלמק והמאכל האהוב עליו הוא פיצה.

פתחו וממשו תוכנית המנהלת את מציאת קרוב המשפחה של בונבון. התוכנית תקלוט את שמו של הטוען לקרבה, את הכוכב שהוא בא ממנו ואת המאכל האהוב עליו.

אם בונבון יזהה קרוב משפחה מבין אחד הטוענים לקרבה, התוכנית תודיע: קרוב המשפחה נמצא, תודה על השתתפותכם. אם לאחר 50 קרובים שנבדקו לא נמצאה התאמה, התוכנית תודיע שהחיפוש הסתיים.

הגדרת המחלקה *חייזר*

הגדרת התכונות

על פי הגדרת הבעיה לעצם מהמחלקה חייזר (כלומר לבונבון ולקרוביו) יש התכונות הבאות:
שם פרטי, שם משפחה, כוכב ומאכל אהוב.
נבחר את המשתנים הבאים לתיאור התכונות:

- **firstName** – מחרוזת. שמו הפרטי של החייזר.
- **lastName** – מחרוזת. שם המשפחה של החייזר.
- **planet** – מחרוזת. כוכב המוצא של החייזר.
- **favoriteFood** – מחרוזת. המאכל האהוב על החייזר.

הגדרת הפעולות

- **פעולה בונה** – הפעולה תקבל כפרמטרים את שם החייזר, את הכוכב שהגיע ממנו ואת המאכל האהוב עליו. הפעולה הבונה תאתחל את תכונות החייזר לפי הפרמטרים שהתקבלו.
- **האם קרוב משפחה** – לכל חייזר הטוען לקרבת משפחה צריך לבדוק אם הוא אכן קרוב משפחה לפי שם משפחתו, לפי הכוכב שהגיע ממנו ולפי המאכל האהוב עליו. כלומר הפעולה מקבלת כפרמטר חייזר אחר הטוען לקרבה – היא תחזיר **true** אם שניהם קרובי משפחה ו-**false** אחרת.

מימוש המחלקה

```
/*
מחלקת חייזר
*/
public class Alien
{
    private string firstName;
    private string lastName;
    private string planet;
    private string favoriteFood;
    public Alien (string firstName, string lastName, string planet,
                                          string favoriteFood)
    {
         this.firstName = firstName;
         this.lastName = lastName;
         this.planet = planet;
         this.favoriteFood = favoriteFood;
    }
    public bool IsRelative(Alien relative)
    {
         if (lastName==relative.lastName && planet==relative.planet
                        && favoriteFood==relative.favoriteFood)
              return true;
         else
              return false;
    }
}// Alien
```

הגדרת הפעולה הראשית

המשימה של הפעולה הראשית היא לקלוט בכל פעם את הטוען לקרבה לבונבון ולבדוק אם הוא אכן קרוב המשפחה של בונבון. אם כן, להציג את הפלט קרוב משפחה נמצא, תודה על השתתפותכם. אחרת להמשיך ולקלוט את הטוען לקרבה הבא בתור.

בחירת משתנים

bonbon – עצם מטיפוס חייזר. מייצג את החייזר בונבון אלף שהגיע ממלמק והמאכל האהוב עליו הוא פיצה.
relative – עצם מטיפוס חייזר. מייצג בכל פעם את הטוען לקרבה משפחתית לבונבון.
firstName – משתנה מטיפוס מחרוזת, מכיל את השם של הטוען לקרבה.
lastName – משתנה מטיפוס מחרוזת, מכיל את שם המשפחה של הטוען לקרבה .
planet – משתנה מטיפוס מחרוזת, מכיל את הכוכב ממנו הגיע הטוען לקרבה.
favoriteFood – משתנה מטיפוס מחרוזת, מכיל את המאכל האהוב על הטוען לקרבה.
foundRelative – משתנה בוליאני, קובע אם החייזר שנבדק הוא קרוב משפחה של בונבון

מימוש

```
/*
  מחלקה ראשית לחיפוש קרוב משפחה של בונבון
*/
using System;
public class AlienFamily
{
    public static void Main()
    {
         string firstName;
         string lastName;
         string planet;
         string favoriteFood;
         bool foundRelative = false;
         Alien bonbon = new Alien("Bonbon","Alf","Melmack","pizza");
         Alien relative;
         int counter = 0;
         while (!foundRelative && counter < 50)
         {
              Console.Write("---Looking for a relative--- ");
              counter ++;
              Console.Write("Enter first name: ");
              firstName = Console.ReadLine();
              Console.Write("Enter last name: ");
              lastName = Console.ReadLine();
              Console.Write("Where do you come from? ");
              planet = Console.ReadLine();
              Console.Write("What is your favorite food? ");
              favoriteFood = Console.ReadLine();
              relative = new
                   Alien(firstName,lastName,planet,favoriteFood);
              foundRelative = bonbon.IsRelative(relative);
         }
         if (foundRelative)
              Console.WriteLine("A relative was found. " +
                                  "Thank you for your participation");
         else
              Console.WriteLine("The search ended with no luck");
    }// Main
}// AlienFamily
```

סוף פתרון בעיה 8

בפתרון בעיה 8 העברנו עצם מסוג חייזר לפעולה IsRelative. כך, ניתן להעביר לפעולות גם עצמים כפרמטרים ולא רק משתנים.

שאלה 11.20

בכל בוקר מוציא מר גרבון גרב ממגירת הגרביים ומתקשה למצוא לו בן-זוג תואם. פתחו תוכנית שתעזור למר גרבון למצוא בן-זוג תואם. התוכנית תקלוט ממר גרבון את הגרב (גודל, צבע ודוגמה) ולאחר מכן תקלוט את בן-הזוג שהוא מנסה להתאים לגרב. התוכנית תסתיים כאשר יימצא גרב תואם. הגדירו מחלקה בשם "גרב" בעלת התכונות גודל, צבע ודוגמה. הגדירו פעולה במחלקה

שתקבל גרב אחר ותחליט אם הגרביים הם זוג. עליכם להחליט לפי שיקול דעתכם אילו טיפוסי נתונים מתאימים לתכונות גודל, צבע ודוגמה.

שאלה 11.21

הגדירו מחלקה המגדירה כיתה. תכונות הכיתה הן מספר התלמידים בכיתה ושם המורה. הגדירו לכיתה פעולה בשם "איחוד": הפעולה תקבל כפרמטר עצם אחר מהמחלקה כיתה ו"תאחד" אותו עם הכיתה הנוכחית. איחוד כיתה עם כיתה אחרת נעשה באמצעות עדכון מספר התלמידים בכיתה הנוכחית לסכום התלמידים בשתי הכיתות ושרשור שם המורה של הכיתה שהתקבלה כפרמטר לשם המורה של הכיתה הנוכחית. נתוני הכיתה שהתקבלה כפרמטר לא ישתנו.

בעיה 9

מטרת הבעיה ופתרונה: העברת עצם כפרמטר לפעולה – העמקה.

ניתן להגדיר חשבון בנק באמצעות מחלקה הכוללת את התכונות הבאות: שם בעל החשבון והיתרה בחשבון. הפעולות שניתן לבצע בחשבון בנק הן: הפקדה, משיכה והעברה. פעולת ההעברה מקבלת כפרמטר את חשבון הבנק שרוצים אליו להעביר כסף ואת סכום הכסף להעברה.

את החשבון מאתחלים באמצעות פעולה בונה המקבלת את שם בעל החשבון ואת היתרה ההתחלתית הנמצאת בחשבון.

פתחו וממשו את המחלקה חשבון בנק.

הגדרת המחלקה *חשבון בנק*

הגדרת התכונות

על פי הגדרת הבעיה לעצם מהמחלקה חשבון בנק יש התכונות הבאות: שם בעל החשבון ויתרה. נבחר את המשתנים הבאים לתיאור התכונות:

- **name** – מחרוזת המייצגת את שם בעל החשבון.
- **balance** – משתנה מטיפוס ממשי המייצג את היתרה בחשבון.

הגדרת הפעולות

- **פעולה בונה** – הפעולה הבונה תקבל כפרמטרים את שם בעל החשבון ואת היתרה. הפעולה הבונה תאתחל את התכונות לפי הפרמטרים שהתקבלו.
- **משיכה** – הפעולה תקבל כפרמטר את סכום הכסף שרוצים למשוך מהחשבון ותעדכן את היתרה בהתאם.
- **הפקדה** – הפעולה תקבל כפרמטר את סכום הכסף שרוצים להפקיד לחשבון ותעדכן את היתרה בהתאם.
- **העברה** – פעולה המעבירה כסף לחשבון אחר. הפעולה תקבל כפרמטר עצם המייצג את החשבון שרוצים אליו להעביר כסף ואת סכום הכסף שיש להעביר.
- **החזר יתרה** – הפעולה תחזיר את היתרה בחשבון.

מימוש המחלקה

```
/*
 מחלקת חשבון בנק
*/
public class BankAccount
{
    private double balance;
    private string name;
    //פעולה בונה
    public BankAccount(string name, double balance)
    {
         this.name = name;
         this.balance = balance;
    }
    //הפקדה
    public void Deposit(double amount)
    {
         balance = balance + amount;
    }
    //משיכה
    public void Withdraw(double amount)
    {
         balance = balance - amount;
    }
    //העברה
    public void Transfer(BankAccount otherAccount, double amount)
    {
         Withdraw(amount);
         otherAccount.Deposit(amount);
    }
    //החזר יתרה
    public double GetBalance()
    {
         return balance;
    }
}// class BankAccount
```

קטע הקוד הבא מתאר העברה של 100 שקלים מחשבון ba1 לחשבון ba2

```
BankAccount ba1 = new BankAccount("ba1", 500);
BankAccount ba2 = new BankAccount("ba2", 200);
//נוכל להעביר 100 שקלים מחשבון 1 לחשבון 2
ba1.Transfer(ba2,100);
```

כלומר כעת בחשבון ba1 היתרה היא 400 שקלים ובחשבון ba2 היתרה היא 300 שקלים. כיצד? אמנם בשפת C# כאשר מעבירים פרמטר מטיפוס פשוט לפעולה ומשנים אותו, ערכו של המשתנה שהועבר כפרמטר אינו משתנה אך כאשר מעבירים עצם כפרמטר לפעולה, עוברת למעשה **ההפניה** אל העצם. באמצעות הפניה זו ניתן לבצע פעולות על העצם שהועבר כפרמטר ובכך לשנות אותו.

סוף פתרון בעיה 9

נסתכל על יישום הפעולה Transfer במחלקה "חשבון בנק". הפעולה מקבלת פרמטר מסוג חשבון בנק ומפעילה עליו את הפעולה Deposit עם סכום הכסף שהתקבל אף הוא כפרמטר. בניגוד לטיפוס פשוט (כפי שראינו במחלקת המחשבון), כאשר מעבירים עצם כפרמטר, עוברת ההפניה אל העצם (הפניה אל המקום בזיכרון ששם שמור העצם) ולא עותק של העצם. מסיבה זו, פעולות המופעלות על הפרמטר מופעלות על העצם עצמו ולא על עותק של העצם.

שאלה 11.22

במשחק "הדִיגֶר" קיים יצור בשם דיגר והוא אוסף ומאבד יהלומים.

א. כתבו מחלקה המגדירה דיגר המכיל את התכונה מספר יהלומים, את פעולת הגישה המחזירה את מספר היהלומים, ואת הפעולות "אסוף יהלום" ו"אבד יהלום". הפעולה "אסוף יהלום" מוסיפה 1 למניין היהלומים של הדיגר והפעולה "אבד יהלום" מפחיתה 1 ממספר היהלומים שיש לדיגר. כל דיגר מתחיל את חייו עם עשרה יהלומים.

ב. כעת יכול הדיגר לאכול דיגרים אחרים. כאשר דיגר אוכל דיגר אחר הוא מקבל (אוסף) את כל היהלומים שהיו לדיגר שאכל. הדיגר שנאכל מאבד את כל היהלומים שלו. הגדירו את פעולת "אכול דיגר".
הדרכה: פעולה זו תקבל דיגר כפרמטר.

שאלה 11.23

נתונה המחלקות Doubler ו-DoublerTest הבאות:

```
using System;
public class Doubler
{
    private double num;
    public Doubler(double num)
    {
         this.num = num;
    }
    public double GetNum()
    {
         return num;
    }
    public void DoubleNum()
    {
         num = num * 2;
         Console.WriteLine("Doubled = {0}", num);
    }
    public void DoubleIt(Doubler it)
    {
         it.DoubleNum();
         Console.WriteLine("Doubled: it = {0}", it.GetNum());
    }
}// Doubler

public class DoublerTest
{
    public static void Main()
    {
         Doubler doubler1 = new Doubler(5);
         Doubler doubler2 = new Doubler(4);
```

```
        Console.WriteLine("doubler1 is {0}", doubler1.GetNum());
        doubler1.DoubleNum();
        Console.WriteLine("doubler1 after DoubleNum is {0}",
                                                  doubler1.GetNum());
        Console.WriteLine("doubler2 is {0}", doubler2.GetNum());
        doubler1.DoubleIt(doubler2);
        Console.WriteLine("doubler2 after DoubleIt is {0}",
                                                  doubler2.GetNum());
    } // Main
}//DoublerTest
```

מה הפלט שיוצג בסוף התוכנית? השלימו:

```
doubler1 after DoubleNum is  _________
doubler2 after DoubleIt is _________
```

בעיה 10

מטרת הבעיה ופתרונה: הצגה של החזרת מערך מפעולה

מוכר בחנות מקבל כבונוס 10% מסכום שלוש המכירות הגבוהות ביותר שמכר באותו חודש. כתבו תוכנית הקולטת את ערכה הכספי של כל מכירה ומכירה שהמוכר מכר בחודש מסוים. התוכנית תציג את שלוש המכירות הגבוהות ביותר ואת סכום הבונוס.

כתבו מחלקה המגדירה מוכר. התכונות של מוכר הם שמו ושלוש המכירות הגבוהות ביותר שמכר. הפעולות המוגדרות עבור מוכר הן: הוספת מכירה, פעולה לחישוב הבונוס ופעולה להחזרת שלוש המכירות הגבוהות ביותר.

הגדרת המחלקה *מוכר*

נזדקק למחלקה המגדירה מוכר.

הגדרת התכונות

על פי הגדרת הבעיה, לעצם מהמחלקה מוכר יהיו התכונות: שם ושלוש המכירות הגבוהות ביותר באותו החודש. בהתאם לכך נבחר את המשתנים הבאים לתיאור התכונות:

- **name** – משתנה מטיפוס מחרוזת, מייצג את שם המוכר.
- **bonusSales** – מערך מטיפוס ממשי, מכיל את שלוש המכירות הגבוהות ביותר.

הגדרת הפעולות

- **פעולה בונה** – הפעולה הבונה תקבל כפרמטר את שמו של המוכר ותאתחל אותו. בנוסף לכך, הפעולה תקצה את מערך המכירות ותאפס אותו.
- **AddSale** – הפעולה תקבל כפרמטר ערך ממשי המייצג מכירה. הפעולה תבדוק אם מכירה זו מועמדת להיות אחת משלוש המכירות הגבוהות בַּחודש, אם כן ערכו יישמר במערך bonusSales.
- **CalculateBonus** – הפעולה איננה מקבלת פרמטרים והיא מחזירה את סכום הבונוס שהמוכר זכאי לו לפי הנוסחה: 10% מסכום המכירות במערך bonusSales.
- **GetBonusSales** – הפעולה מחזירה את המערך של שלוש המכירות הגבוהות בחודש.

מימוש המחלקה

```
/*
המחלקה מוכר
*/
public class Seller
{
    private double[] bonusSales;
    private string name;
    //הפעולה הבונה
    public Seller(string name)
    {
         this.name = name;
         bonusSales = new double[3];
         for (int i = 0; i < bonusSales.Length; i++)
              bonusSales[i] = 0.0;
    }
    //פעולה להוספת מכירה
    public void AddSale(double sale)
    {
         //מציאת המכירה הקטנה ביותר
         double min = bonusSales[0];
         int minIndex = 0;
         for (int i = 1; i < bonusSales.Length; i++)
         {
              if (min > bonusSales[i])
              {
                   min = bonusSales[i];
                   minIndex = i;
              }
         }
         //אם המכירה הנוכחית גדולה ממנה מעדכנים את המערך
         if (sale > min)
              bonusSales[minIndex] = sale;
    }
    //פעולה המחזירה את הבונוס שהמוכר זכאי לו
    public double CalculateBonus()
    {
         double bonus = 0;
         for (int i = 0; i < bonusSales.Length; i++)
              bonus = bonus + bonusSales[i] * 0.1;
         return bonus;
    }
    //פעולה המחזירה את מערך שלוש המכירות הגבוהות של החודש
    public double[] GetBonusSales()
    {
         // העתקת מערך המכירות והחזרתו
         double[] sales = new double[bonusSales.Length];
         for (int i = 0; i < bonusSales.Length; i++)
              sales[i] = bonusSales[i];
         return sales;
    }
} // Seller
```

שימו ♥: הפעולה GetBonusSales מחזירה **עותק** של המערך bonusSales ולא את המערך עצמו. הסיבה לכך היא שמערך ב-C# ממומש באמצעות הפניה, ולכן אם נחזיר את ההפניה למערך bonusSales (**return** bonusSales;), תוכל הפעולה הראשית לשנות את תוכן המערך בצורה ישירה באמצעות הפניה זו. הדבר אינו רצוי משום שהמערך bonusSales הוגדר כתכונה פרטית של המוכר, ואנו רוצים לוודא שערכיו יתעדכנו בצורה מבוקרת, אך ורק לפי האלגוריתם שקבענו בפעולה AddSale. בצורה זו אנחנו מגנים על הנתונים השמורים בעצם מוכֵר מפני שינויים מבחוץ. כל שינוי שיתבצע על המערך המוחזר לא ישפיע על נתוני המכירות בעצם מוכֵר.

הגדרת הפעולה הראשית

הפעולה הראשית תיצור עצם מסוג מוכר ותקלוט את נתוני המכירות שלו בחודש מסוים. סיום הנתונים יסומן במכירה שערכה הוא 0. התוכנית תחשב ותציג את הבונוס ואת שלוש המכירות הגבוהות ביותר של המוכר:

```
/*
        Seller מחלקה ראשית המשתמשת במחלקה
*/
using System;
public class SellerBonus
{
    public static void Main()
    {
         // יצירת עצם מסוג מוכר
         Seller seller = new Seller("selli");

         // קלט ועדכון המכירות
         double sale;
         Console.Write("Enter a sale: ");
         sale = double.Parse(Console.ReadLine());
         while (sale > 0.0)
         {
              seller.AddSale(sale);
              Console.Write("Enter a sale, Enter 0 to end: ");
              sale = double.Parse(Console.ReadLine());
         }
         // חישוב בונוס והצגת פלט
         double bonus;
         bonus = seller.CalculateBonus();
         Console.WriteLine("The seller bonus is {0}", bonus);
         double[] sales;
         sales = seller.GetBonusSales();

         // output the sales
         Console.Write("The bonus Sales are: ");
         for (int i = 0; i < sales.Length; i++)
              Console.Write(" {0} ", sales[i]);
         Console.WriteLine();
    } // Main
} // class SellerBonus
```

סוף פתרון בעיה 10

שאלה 11.24

בתחרות קפיצה לרוחק כל משתתף קופץ 5 קפיצות. כתבו מחלקה המגדירה קופץ לרוחק. עצם מהמחלקה ישמור את שמו של הקופץ ואת תוצאות הקפיצות שלו.
פתחו וממשו תוכנית לאימון בקפיצה לרוחק. התוכנית תקלוט את שמו של הקופץ. לאחר מכן תבקש התוכנית מהקופץ להכניס בכל פעם את תוצאת קפיצתו. (שימו לב שתוצאה של קפיצה לרוחק היא מספר ממשי, למשל 3.15 מטר).
התוכנית תציג כפלט את שמו של הקופץ, את קפיצותיו ואת ממוצע הקפיצות.

שאלה 11.25

כתבו מחלקה המגדירה ספר טלפונים אלקטרוני (כמו בטלפון נייד). ספר טלפונים מכיל מערך של מחרוזות. כל מחרוזת מורכבת משם וממספר טלפון. הניחו שהמערך ממוין לפי השמות. הגדירו פעולה המקבלת מחרוזת ומחזירה מערך המכיל את כל המחרוזות שמתחילות במחרוזת זו.
לדוגמא: נניח שבמערך המחרוזות 3 מחרוזות:
"דונלד 4444"
"מיקי 1234"
"מיני 998877"
כאשר הפרמטר לפעולה יהיה "מיק" תחזיר הפעולה מערך המכיל את המחרוזות:
"מיקי 1234"
כאשר הפרמטר לפעולה יהיה "מ" תחזיר הפעולה מערך המכיל את המחרוזות:
"מיקי 1234"
"מיני 998877"

סיכום

בפרק זה למדנו כיצד להגדיר מחלקות חדשות ולהשתמש בעצמים ממחלקות אלו.

הגדרה של **מחלקה** היא למעשה הגדרה של טיפוס חדש. הגדרת **מחלקה** כוללת הגדרה של תכונות ושל פעולות עבור **עצמים** של המחלקה, כלומר עבור משתנים מטיפוס המחלקה.

- **תכונות של עצם** הן משתנים מטיפוסים כלשהם (כולל מערכים ואף עצמים כגון מחרוזות), המאפיינים את העצם. בדרך כלל נגדיר את התכונות כ**פרטיות (private)** על מנת להגן עליהן מפני שינוי לא מבוקר מבחוץ וכדי לא לחשוף את אופן הייצוג הפנימי של העצם.
- **פעולות של עצם** הן פעולות המשויכות לעצם, והן פועלות בדרך כלל על תכונות העצם. את הפעולות נגדיר בדרך כלל כ**ציבוריות (public)**. **פעולות של עצם** יכולות לקבל פרמטרים ולהחזיר ערך.
- **פרמטרים** לפעולה יכולים להיות משתנים מכל טיפוס שהוא. ראינו שכאשר משתנים מטיפוסים פשוטים כגון **int** מועברים כפרמטרים, עובר רק הערך של המשתנה כפרמטר. כלומר שינוי של משתנה כזה בתוך הפעולה **איננו משנה** את ערכו של המשתנה מחוץ לפעולה. לעומת זאת, כאשר מערכים ועצמים מועברים כפרמטרים, עוברת ההפניה אל שטח הזיכרון שהעצם נמצא בו. באמצעות הפניה זו ניתן לשנות את תוכן המערך או העצם מתוך הפעולה. מכיוון שהשינוי מתבצע על המערך או על העצם עצמו ולא על עותק שלו, שינוי כזה תקף גם אחרי סיום הפעולה.

- **החזרת ערך מפעולה** יכולה להיות מטיפוס כלשהו. אופן החזרת הערך דומה לאופן העברת הפרמטרים, כלומר במקרה של ערך מטיפוס פשוט, מוחזר העותק שלו ובמקרה של ערך מטיפוס מערך או עצם, מוחזרת ההפניה אליו. לכן, אם קיימת תכונה שהיא מערך, נשקול להחזיר עותק של המערך ולא הפניה למערך עצמו, כדי שלא תהיה גישה ישירה למערך מבחוץ.
- **פעולה בונה** היא פעולה מיוחדת המופעלת מיד אחרי שמוקצה עבור העצם שטח בזיכרון, באמצעות ההוראה `new`. פעולה בונה מחזירה הפניה לעצם שנוצר, אך בכותרתה אין לכלול טיפוס ערך מוחזר. שמה של הפעולה הבונה חייב להיות זהה לשם המחלקה. כמו כל פעולה, גם פעולה בונה יכולה לקבל פרמטרים. פרמטרים אלה משמשים לאתחל את תכונות העצם.
- **פעולות גישה** הן פעולות המשמשות צינור גישה בטוח אל תכונות העצם מחוץ למחלקה. יש שני סוגים של פעולות גישה: פעולות המחזירות ערך של תכונה (Get) ופעולות המעדכנות ערך של תכונה (Set).
- פעולות של עצם יכולות להתייחס ישירות לתכונות של העצם וכן לקרוא לפעולות אחרות של העצם.
- כדי להתייחס לתכונה ששמה זהה לשם של פרמטר, יש להשתמש בקידומת **this** לפני שם התכונה.
- **יצירת עצם** כוללת את השלבים הבאים: הצהרה על משתנה מטיפוס המחלקה, הקצאת זיכרון ואתחול באמצעות הפעולה הבונה.
- **שימוש בעצם** כולל הפעלת הפעולות הציבוריות שלו.
- ניתן לשמור עצמים במערך, או במילים אחרות מערך שכל תא בו מפנה לעצם.
- מחלקה המכילה רק פעולות ללא תכונות נקראת "מחלקת שירות". היא מכילה אוסף של שירותים (פעולות) שאינם פועלים על עצם כלשהו. פעולות אלה יכולות לקבל פרמטרים ולהחזיר ערך אך הן לא פועלות על תכונות של עצם. פעולות אלה נקראות פעולות מחלקה (**class methods**) או פעולות סטטיות (**static methods**).

סיכום מרכיבי שפת #C שנלמדו בפרק 11

הגדרת מחלקה בשפת #C

הגדרת מחלקה נכתבת באופן הבא:

```
public class שם המחלקה
{
        // הגדרת התכונות
        .
        .
        // הגדרת הפעולות
        .
        .
}
```

- המילה השמורה `public` מציינת שהמחלקה פתוחה לשימוש ציבורי. כלומר שמחלקות אחרות יכולות ליצור עצמים מטיפוס המחלקה.

- כדי להגדיר מחלקה נצהיר עליה באמצעות המילה השמורה **class**. לאחר ציון המילה **class** נציין את שם המחלקה. מקובל להתחיל שם מחלקה באות גדולה. אם שם המחלקה מורכב מכמה מילים, הן נכתבות צמודות ללא רווח, באותיות קטנות, וכל מילה מתחילה באות גדולה.

הגדרת תכונות בשפת C#

הגדרת תכונות נכתבת בדומה להצהרה על משתנים :

```
הרשאת-הגישה טיפוס-התכונה שם-התכונה
private double length;
```

- הרשאת הגישה **private** מציינת שהתכונה היא פרטית, כלומר היא מוכרת רק בתוך תחום המחלקה, ורק מתוכו ניתן להתייחס אליה.

הגדרת פעולות בשפת C#

הגדרת פעולה נכתבת באופן הבא :

```
הרשאת-הגישה טיפוס-הערך-המוחזר שם-הפעולה (רשימת פרמטרים)
{
      גוף הפעולה
}
```

לדוגמה :

```
public double GetLength()
{
      return length;
}
```

- הרשאת הגישה **public** מציינת שהפעולה היא ציבורית, כלומר היא מוכרת מחוץ לתחום המחלקה.
- הטיפוס-המוחזר מציין את טיפוס הערך שתחזיר הפעולה.
- במקרה של פעולה שאינה מחזירה ערך יש לרשום **void** במקום הערך החוזר.
- רשימת הפרמטרים מופיעה בסוגריים והיא כוללת זוגות המורכבים מטיפוסים ומשמות משתנים המופרדים בפסיקים.
- המילה השמורה **retrun** מציינת את הוראת החזרה מהפעולה. במקרה שהפעולה מחזירה ערך, יש לכלול ב-**return** ערך או ביטוי מהטיפוס המוחזר.
- פעולה סטטית היא פעולה השייכת למחלקה ולא לעצם כלשהו. על מנת להגדיר פעולה סטטית נוסיף את המילה **static** להגדרת הפעולה, למשל:

```
public static int DigitSum (int num)
```

הפעולה הבונה

הפעולה הבונה מוגדרת באופן הבא :

```
הרשאת-הגישה שם-המחלקה (רשימת פרמטרים)
{
}
```

- הפעולה הבונה משמשת לאתחול תכונות העצם.
- שפת C# מאתחלת באופן אוטומטי תכונות שלא אותחלו במפורש.

- ♦ כאשר לא מגדירים פעולה בונה כלשהי, שפת C# מספקת **כברירת מחדל** פעולה בונה ריקה חסרת פרמטרים.

שימו ♥: למען קריאות המחלקה חשוב מאוד לצרף להגדרתה הערות המתארות את המחלקה, את תכונותיה ואת פעולותיה.

שימוש בעצמים

מחלקה מגדירה טיפוס, וכדי להשתמש בו ניצור עצם מטיפוס המחלקה. יצירת עצם (למשל מתוך הפעולה הראשית) נעשית באופן הבא:

```
Time tm = new Time();
```

- ♦ יצירת העצם כוללת הצהרה על העצם, הקצאת מקום בזיכרון באמצעות ההוראה **new** ואתחול תכונות העצם באמצעות הפעולה הבונה.
- ♦ הפעלת פעולות העצם נעשית בסימון הנקודה באופן הבא:

```
tm.SetTime(8,30);
```

- ♦ יש לשים לב להתאמה בין אופן השימוש בפעולה של העצם לבין כותרת הפעולה: אם הפעולה מקבלת פרמטרים יש להקפיד לספק פרמטרים לפי הסדר שבו הם מופיעים בכותרת הפעולה. יש להקפיד על התאמה של טיפוס הפרמטרים המועברים לטיפוסים המפורטים בכותרת הפעולה; אם הפעולה מחזירה ערך יש להקפיד על התאמה של טיפוס הערך המוחזר לטיפוס הביטוי שמשולב בו הערך.
- ♦ זימון פעולה סטטית נעשה ישירות דרך שם המחלקה:

```
שם המחלקה.שם הפעולה(פרמטרים)
```