פרק 2
עצמים

עד לכאן הכרתם כבר טיפוסי ערכים פשוטים המוגדרים בשפה כמו **int** או **double**, וכן יצרתם משתנים היכולים להכיל ערכים מטיפוסים אלו.

בפרק זה נכיר מושג חדש: **עצם** (**Object**). יש קווי דמיון בין עצם למשתנה, אך יש ביניהם גם הבדלים. לעצם יש מצב פנימי היכול להשתנות במהלך התוכנית – בזה הוא דומה למשתנה. אולם בעוד שמצבו של משתנה הוא ערך יחיד מטיפוס מסוים, מצבו של העצם מורכב, והוא יכול להכיל ערכים מטיפוסים שונים. ההבדל היותר משמעותי הוא, שלעצם יש פעולות שהוא יכול לבצע בהתאם להודעות המגיעות אליו. למשתנה אין פעולות. לבסוף, משתנים אינם קיימים מראש אלא נוצרים בתוכנית וכך גם עצמים. וגם כאן יש הבדלים: משתנה נוצר אוטומטית מרגע הצהרתו, פרמטר של פעולה (אף הוא משתנה) נוצר אוטומטית כאשר מפעילים את הפעולה. בפסקל וגם ב-C ניתן לבקש בתוכנית במפורש יצירת משתנה. יצירת עצמים לעולם אינה אוטומטית, אלא על ידי דרישה המופיעה בתוכנית.

משתנה מטיפוס פשוט כמו **int** ניתן לייצר על ידי הצהרה: **int** x = 3;

ביצוע הצהרה זו מקצה מקום בזיכרון שגודלו נקבע על ידי הטיפוס. מבחינת הפעולות, על כל המשתנים מכל טיפוס שהוא ניתן לבצע בדיוק אותן פעולות: קריאת הערך המאוחסן בהם והצבת ערך. לעומת אחידות זו, אפשרי וסביר שבתוכנית יהיו עצמים השונים זה מזה במבנה שלהם ובפעולות שהם יכולים לבצע. כדי לייצר עצם יש להגדיר את הייצוג הרצוי, וכן לכתוב את הקוד לפעולותיו. ההגדרה של ייצוג העצם ושל הפעולות שהוא יכול לבצע נקראת **מחלקה** (**Class**). לאחר שנכתבה מחלקה, ניתן לייצר ממנה עצמים רבים. לכולם יהיה המבנה המוגדר במחלקה, וכל אחד מהם יוכל לבצע את הפעולות המוגדרות בה. הגדרת מחלקה היא הגדרה של טיפוס חדש באותו השם.

כך למשל: לאחר שנכתבה המחלקה Bucket, קיים טיפוס בשם זה, דלי. כל עצם הנוצר ממחלקה זו הוא מטיפוס Bucket. ההצהרה: Bucket b;

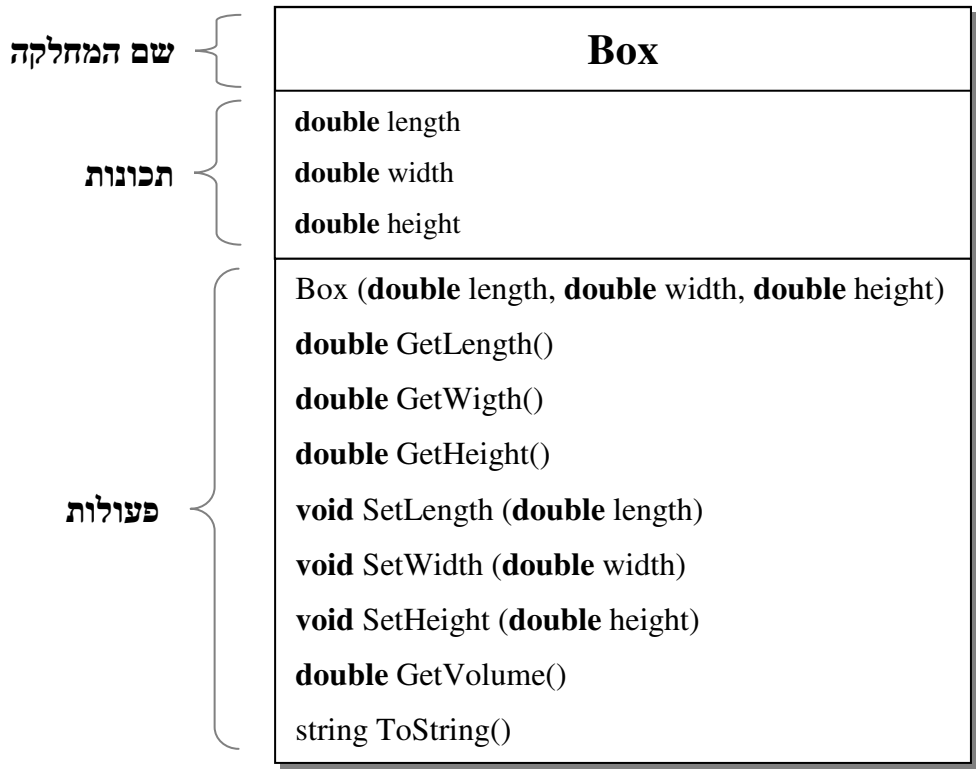
יוצרת משתנה מטיפוס Bucket שיוכל להכיל עצם כלשהו מטיפוס זה.

לכל עצם יש תכונות ופעולות. **תכונות** (**attributes**) הן משתנים פנימיים של העצם המאפיינים את מצבו. כשמגיעה הודעה ממחלקה ליצירת עצם, נבנה עצם חדש ובשעת בנייתו מוצבים בתכונותיו ערכים התחלתיים. למשל: לעצם מטיפוס "קופסה" יש אורך, רוחב וגובה (ייתכן ויש לו תכונות נוספות). תכונות הקופסה מקבלות ערכים בזמן בניית קופסה מסוימת, לפי החלטת הכותב. לתכונות של עצמים שונים מאותו טיפוס יכולים כמובן להיות ערכים שונים, כשם שלקופסאות שונות יש ממדים שונים. יכול להיות גם שלעצמים שונים יהיו ערכי תכונות שווים ובכל זאת הם יהיו מופעים שונים של אותה מחלקה. ערכי התכונות של עצמים יכולים להשתנות במהלך תוכנית.

ה**פעולות** (**methods**) מכירות את תכונות העצם ויכולות לאחזר את ערכיהן או לשנותן. למשל ניתן להגדיר עבור עצם מטיפוס קופסה את הפעולות הבאות: אחזור גובה הקופסה, שינוי המימדים של הקופסה, חישוב נפח הקופסה וחישוב שטח הפנים שלה. (ייתכנו פעולות נוספות). העצם מבצע פעולה כאשר הוא מקבל הודעה מתאימה.

מקובל להציג את המחלקה בעזרת תרשים UML כמתואר באיור הבא. תרשים זה מחולק לשלושה חלקים: שם המחלקה, תכונות ופעולות.

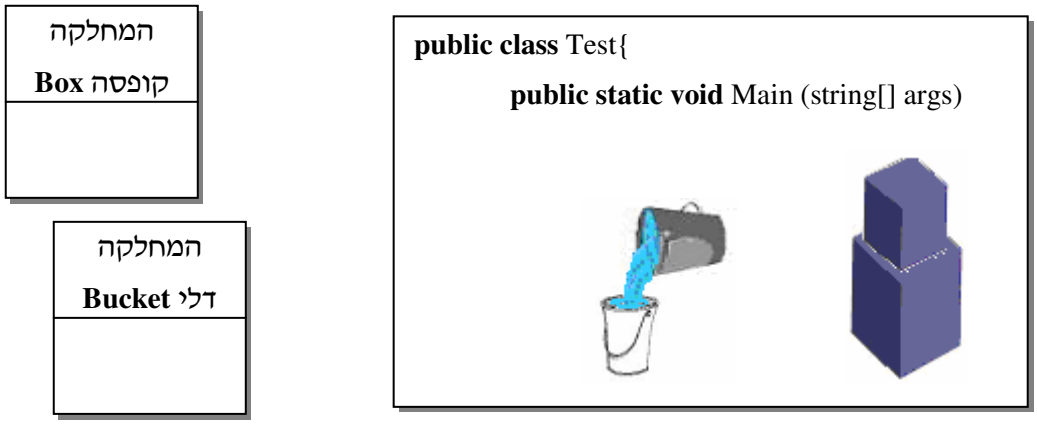
להלן תרשים UML של המחלקה Box:



דוגמה נוספת של עצם היא: "דלי". דליים נבדלים זה מזה בקיבולתם, ולכן סביר שתהיה לדלי תכונה בשם "קיבולת". הדליים נבדלים גם בצבעם, בסוג החומר שממנו הם עשויים, במחירם וכדומה. בחירת התכונות של עצמים היא תהליך של הפשטה: אנו מייצגים יישויות מסוימות תוך הדגשת תכונות מסוימות והתעלמות מאחרות, שאינן נראות לנו חשובות ליישום שאנו בונים. לכן נבחר כתכונות הדלי רק את התכונות הנחוצות לתוכנית המסוימת בה אנו עוסקים. אם למשל, התוכנית עוסקת במכירת דליים, אז מחיר היא תכונה משמעותית ונחוצה. תכונה נוספת של דלי, כמות המים שבו, היא תכונה שיכולה לעניין אותנו אם התוכנית עוסקת במילוי דליים וריקונם.

באופן כללי, תוכנית מכילה הגדרות של מחלקות, ובמהלך ביצועה נוצרים עצמים מהמחלקות. האיור הבא מדגים מבנה כללי של תוכנית. התוכנית משתמשת בשני טיפוסים: קופסה ודלי. לצורך הגדרת טיפוסים אלו נכתבו שתי מחלקות. בפעולה הראשית שבתוכנית נוצרו עצמים

מטיפוסי המחלקות. עצמים אלו מפעילים פעולות שונות ומעבירים הודעות ביניהם כדי למלא משימה כלשהיא.



באיור זה Test היא המחלקה הראשית. היא מכילה את קטע הקוד היוצר עצמים מהמחלקות קופסה ודלי ומעביר הודעות אל העצמים וביניהם במטרה שיבצעו פעולות שונות.
תהליך של תכנות מונחה עצמים כולל שני שלבים:

1. כתיבת מחלקות המגדירות טיפוסי עצמים חדשים.
2. כתיבת תוכנית היוצרת עצמים מטיפוס המחלקות ומשתמשת בהם.

בפרק זה נלמד כיצד ליצור עצמים ממחלקות שכבר הוגדרו ולהשתמש בהם.
בפרק הבא נלמד כיצד לכתוב מחלקות.

ממשק

לכל מחלקה מצורף מסמך המפרט כיצד להשתמש בה. למסמך כזה קוראים **ממשק** (**interface**). בממשק מוגדרת הדרך, או הדרכים, לייצור עצם מטיפוס המחלקה, ומפורטות בו הפעולות שעצם יכול לבצע.

עבור כל פעולה מופיעה ה**כותרת** שלה ובה: שם הפעולה, טיפוסי הפרמטרים שהיא מקבלת וכן טיפוס הערך שהיא מחזירה. כמו כן מכיל הממשק תיאור מילולי של הפעולה. על מנת להשתמש במחלקה אין צורך לדעת כיצד היא ממומשת, מספיק לדעת באיזה אופן משתמשים בה. עיקרון זה נקרא: **הפרדה בין ממשק למימוש**.

מכיוון שהיכולת להשתמש באופן נכון ומדויק במחלקה תלויה לחלוטין ברמת ההסבר והדיוק של הממשק, מובן מדוע חשוב ביותר להקפיד על **תיעוד** מלא ומדויק של קוד המחלקה. התיעוד הפנימי צריך להפוך לטקסט של הממשק ולכן ברור שיש להקפיד על תיאור מדויק של הפעולה, הפרמטרים שהיא מקבלת, מקרי קצה שונים וערכי ההחזרה המדויקים שלה, כמו גם הנחות יסוד

המבטיחות את הפעילות התקינה של המחלקה - הדבר יבטיח שימוש טוב ותקין על ידי המשתמש החיצוני במחלקה.

המושגים הללו: ממשק, מימוש, תיעוד וההפרדה בין ממשק למימוש אינם ייחודיים לתכנות, הם משמשים אותנו בטבעיות בתחומים רבים.
לדוגמה כאשר אנו קונים טלפון סלולרי חדש, אין צורך שנדע מהו המבנה הפנימי שלו. מספיק שנקבל חוברת הדרכה המפרטת באיזו אופן מפעילים את כל האפשרויות שמציע המכשיר.
כאשר אנו משתמשים בתוכנה מסוימת שנכתבה עבורנו, למשל Word, אין צורך לדעת איך היא נכתבה וכיצד היא ממומשת, מספיק שנדע היכן להקליק עם העכבר ובאילו תפריטים לחפש את מה שנחוץ לנו.

לפניכם הממשק של המחלקה "קופסה" – Box.
נדגיש: מכיוון שהפעולות מתבצעות על ידי העצם עצמו, אין צורך להזכיר את העצם בממשק. העצם אינו פרמטר של הפעולה אלא המפעיל שלה.

כותרת הפעולה	תיאור הפעולה
Box (**double** length, **double** width, **double** height)	הפעולה הבונה של המחלקה קופסה. הפעולה מקבלת 3 פרמטרים: אורך, רוחב וגובה ובונה קופסה חדשה בהתאם לפרמטרים אלו
double GetLength()	הפעולה מחזירה את האורך של הקופסה
double GetWigth()	הפעולה מחזירה את הרוחב של הקופסה
double GetHeight()	הפעולה מחזירה את הגובה של הקופסה
void SetLength (**double** length)	הפעולה משנה את האורך על פי הערך המתקבל כפרמטר
void SetWidth (**double** width)	הפעולה משנה את הרוחב על פי הערך המתקבל כפרמטר
void SetHeight (**double** height)	הפעולה משנה את הגובה על פי הערך המתקבל כפרמטר
double GetVolume()	הפעולה מחשבת את הנפח של הקופסה
string ToString()	הפעולה מחזירה מחרוזת המתארת את הקופסה

הפעולה הבונה

פעולה בונה (**constructor**) מגדירה דרך לייצר עצם מטיפוס המחלקה. השם של פעולה זו הוא כשם הטיפוס אותו רוצים לייצר וזהו גם שמו של הערך המוחזר בסיום הפעולה. שימו לב: הפעולה הבונה אינה מופעלת על ידי עצם מסוים אלא מתוך המחלקה כדי לייצר עצם חדש. שאר הפעולות מיועדות לביצוע על ידי עצמים שנוצרו מהמחלקה.

כך מתוארת הפעולה הבונה בממשק של המחלקה "קופסה" - Box:

תיאור הפעולה	כותרת הפעולה
הפעולה הבונה של המחלקה קופסה. הפעולה מקבלת 3 פרמטרים: אורך, רוחב וגובה ובונה קופסה חדשה בהתאם לפרמטרים אלו	Box (**double** length, **double** width, **double** height)

ניתן ליצור קופסאות בעזרת הפעולה הבונה הזאת. זימון הפעולה הבונה מתבצע בעזרת המילה השמורה **new**.

על מנת ליצור קופסה באורך 5 , ברוחב 3.2 ובגובה 10, עלינו לזמן את הפעולה הבונה באופן הבא:

new Box (5, 3.2, 10)

על מנת ליצור קופסה באורך 3, ברוחב 2 ובגובה 1.3, עלינו לזמן את הפעולה הבונה עם פרמטרים מתאימים:

new Box (3, 2, 1.3)

מקובל לקרוא לעצם שיצרנו מהמחלקה, **מופע** (**instance**) של המחלקה.

הפניות

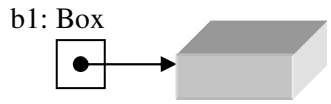
עד כה הכרנו משתנים היכולים להכיל ערכים פשוטים, כגון מספרים, אבל משתנה יכול גם "להכיל" עצם. בפועל המשתנה מכיל **הפניה** (**reference**) אל העצם. כאשר מוגדר משתנה מטיפוס מחלקה מסוימת אזי הוא יכול להכיל הפניות לעצמים שהם מופעים של המחלקה, כלומר: עצמים מטיפוס המחלקה.

לדוגמה ההגדרה: `Box b1;`

יוצרת משתנה b1 מטיפוס Box. ערכו של b1 אינו ידוע ועלינו לאתחל אותו.

ניצור מופע של Box ונציב את ההפניה אליו בתוך המשתנה: `b1 = new Box (5, 3, 2);`

עכשיו ההפניה פונה לעצם מטיפוס Box:



אפשר לאחד את הגדרת המשתנה והצבת העצם בו, ולכתוב:

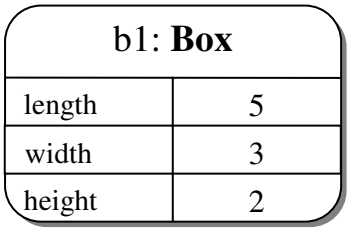
```
Box b1 = new Box (5, 3, 2);
```

ההכרזה באגף שמאל יוצרת משתנה מטיפוס Box.

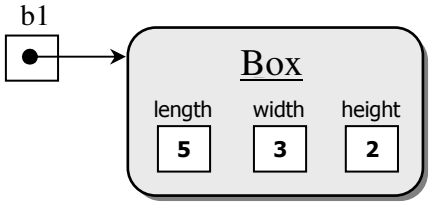
באגף ימין נוצר עצם מטיפוס Box שתכונותיו מאותחלות על פי ערכי הפרמטרים. באגף זה Box הוא שם הפעולה הבונה וגם טיפוס העצם שהפעולה מחזירה. ביצוע הפקודה יוצר אם כן משתנה ועצם, ומציב במשתנה את ההפניה לעצם.

באופן שכזה גם נתנו "שם" לעצם החדש שנוצר: b1. מעתה נוכל לדבר על העצם b1. יכול להיות שלאותו מופע תהיה הפניה על ידי כמה משתנים כך שיהיו לו "שמות שונים".

מקובל גם להציג את העצם מטיפוס Box שערכיו פורטו לעיל, וההפניה אליו היא b1, בעזרת אחד מהסרטוטים הבאים. כך אנו מציגים את כל האינפורמציה שיש בידינו לגבי המופע, באיור מוסכם, באופן מקוצר ותמציתי:



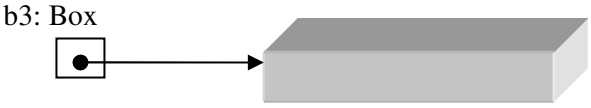
b1: **Box**	
length	5
width	3
height	2



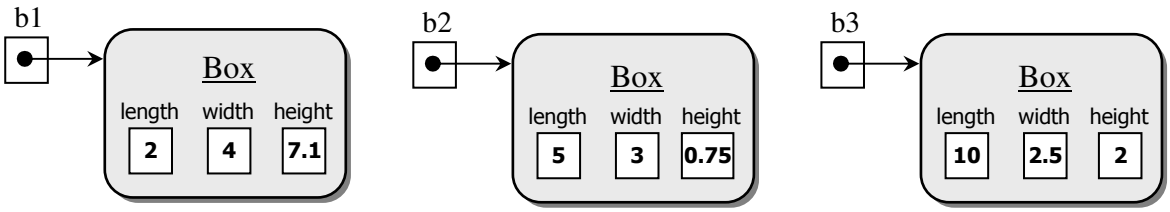
ניצור עתה בפעולה הראשית מספר מופעים של Box:

```
public class Test
{
        public static void Main (string[] args)
        {
                Box b1 =  new Box (2, 4, 7.1);
                Box b2 =  new Box (5, 3, 0.75);
                Box b3 =  new Box (10, 2.5, 2);
        }
}
```

יצרנו שלוש קופסאות שונות שאליהן מפנים שלושה משתנים שונים מטיפוס Box:



בעזרת ה-UML שפת המידול שבה אנו נעזרים (ראו בנספח 1), יראו שלוש הקופסאות כך:



ערכי ברירת מחדל

משתנים מקומיים, הן מטיפוס בסיסי והן הפניות לעצמים, המוגדרים בתוכנית הראשית או בתוך פעולה, אינם מקבלים ערך ברירת מחדל ואנו חייבים לאתחל אותם בערך הרצוי. במידה ולא יאותחלו נקבל שגיאת הידור והתוכנית לא תעבור קומפילציה.

לעומת זאת תכונות של עצמים, יאותחלו במהלך הפעלת הפעולה הבונה. אם לא נכתוב במפורש מהם ערכי האתחול יקבלו התכונות את ערכי ברירת המחדל של השפה. תכונה שהיא ערך מספרי תאותחל לאפס. תכונה שהיא עצם תאותחל להפניה ריקה: **null**, שפירושה שהמשתנה אינו מפנה לאף עצם. את הערך **null** אנו מסמנים באופן הבא: ⫴ , או שאנו מסמנים שהערך במשתנה הוא **null**:

פעולות נוספות

אחרי שהעצם נוצר הוא יכול לבצע את הפעולות המוגדרות בממשק. פעולות יכולות לקבל ערכים כפרמטרים וכן להחזיר ערכים.

כאשר רוצים לברר ערך של תכונה כלשהי בקופסה משתמשים בפעולות המחזירות את ערכי התכונה:

תיאור הפעולה	כותרת הפעולה
הפעולה מחזירה את האורך	**double** GetLength()
הפעולה מחזירה את הרוחב	**double** GetWigth()
הפעולה מחזירה את הגובה	**double** GetHeight()

פעולות אלו אינן מקבלות פרמטרים, אך למרות זאת חובה לכתוב את הסוגריים מיד לאחר שם הפעולה. יש לפעולות ערך החזרה מטיפוס **double**, שהוא טיפוס התכונה המבוקשת.

זימון פעולה נעשה בעזרת שיטת סימון מיוחדת הנקראת **סימון-הנקודה** (**dot notation**). כדי להודיע לעצם להפעיל פעולה, נכתוב את שם ההפניה המפנה אליו, מימין להפניה נכתוב נקודה, מימינה את שם הפעולה המבוקשת ואחריה את ערכי הפרמטרים בסוגריים.

```
Box b1 = new Box (2, 2, 2);
double x = b1.GetLength();
```

בשורה הראשונה יצרנו עצם מטיפוס Box. קופסה מרובעת שרוחבה, אורכה וגובהה הם 2. את ההפניה לעצם הצבנו בתוך משתנה מטיפוס Box הנקרא b1. בשורה השנייה פנינו ל-b1 בשיטת סימון-נקודה כדי להפעיל את הפעולה GetLength(). פעולה זו מחזירה את אורך הריבוע שאותו אנו מציבים למשתנה מטיפוס **double**.

הפעולה הבונה מזומנת באופן שונה, באמצעות המילה השמורה **new**. זו אינה פעולה המתבצעת על ידי עצם אלא פעולה היוצרת עצם חדש ממחלקה.

נדגים זימון של פעולה המחזירה ערך. הזימון מופיע בתוך הפעולה הראשית:

```
public static void Main (string[] args)
{
        Box b1 = new Box (2, 4, 6);
        double x = b1.GetWidth();
        Console.WriteLine (x);
}
```

בשורה הראשונה של הפעולה הראשית יוצרים מופע של Box ומציבים הפניה אליו בתוך משתנה מטיפוס Box הנקרא b1. בשורה השנייה מפעילים את ()GetWidth דרך ההפניה b1. פעולה זו מחזירה את הרוחב של הקופסה. הערך המוחזר מוצב לתוך המשתנה x שהוא מטיפוס **double**. מכיוון שעל פי היצירה של הקופסה, הערך של x הוא 4 , תדפיס הפעולה הראשית בשורה האחרונה את הערך הזה.

יש סוג נוסף של פעולות. פעולות אלו משנות את ערכי התכונות של העצם ואינן מחזירות ערך, לכן אנו מציינים שטיפוס החזרה שלהן הוא ריק על ידי ציון הטיפוס **void**. הפעולה (…)SetLength מקבלת כפרמטר ערך מטיפוס **double** ששמו length. פעולה זו משנה את תכונת האורך של הקופסה להיות הערך החדש שמתקבל כפרמטר.

תיאור הפעולה	כותרת הפעולה
הפעולה משנה את האורך על פי הערך המתקבל כפרמטר	**void** SetLength (**double** length)

בממשק של הקופסה יש פעולות נוספות המשנות ערכים של רוחב הקופסה וגובהה:

תיאור הפעולה	כותרת הפעולה
הפעולה משנה את הרוחב על פי הערך המתקבל כפרמטר	**void** SetWidth (**double** width)
הפעולה משנה את הגובה על פי הערך המתקבל כפרמטר	**void** SetHeight (**double** height)

נראה פעולה נוספת שמבצעת חישוב על התכונות של העצם ומחזירה את תוצאת החישוב:

הפעולה מחשבת את הנפח של הקופסה	**double** GetVolume()

פעולה זו אינה מקבלת פרמטרים וערך החזרה שלה הוא הנפח של הקופסה, מטיפוס **double**.

זימון הפעולה:

```
public static void Main (string[] args)
{
        Box b1 = new Box (2, 4, 6);
        Box b2 = new Box (1, 1, 1);
        double x1 = b1.GetVolume();
        double x2 = b2.GetVolume();
        Console.WriteLine (x1);
        Console.WriteLine (x2);
}
```

? מה יודפס כתוצאה מהרצת הפעולה הראשית?

הפעולה האחרונה בממשק המחלקה "קופסה" מחזירה מחרוזת המתארת את העצם:

הפעולה מחזירה מחרוזת המתארת קופסה	string ToString()

פעולה זו מחזירה מחרוזת המתארת את העצם בצורה עליה החליט המתכנת.

המחרוזת יכולה להיות:

"The length of the box is 2, the width of the box is 4, the height of the box is 6"

אך היא יכולה להיות גם קצרה בהרבה: "A box: 2, 4, 6"

על אופן ההצגה מחליט המתכנת המממש את הפעולה ToString(). פעולה זו משמשת אותנו לרוב לצורך הדפסת עצמים.

זימון הפעולה ToString():

```
public static void Main (string[] args)
{
        Box b1 = new Box (2, 4, 6);
        string str = b1.ToString();
        Console.WriteLine (str);
}
```

בשורה הראשונה בפעולה הראשית נוצר עצם מטיפוס קופסה באורך 2, רוחב 4 וגובה 6. עצם זה מוצב בהפניה b1. בשורה השנייה מופעלת הפעולה ToString() שמחזירה מחרוזת המתארת את

הקופסה (בהתאם לממשק), מחרוזת זו מוצבת בהפניה מטיפוס מחרוזת, הנקראת str. בשורה האחרונה המחרוזת הזו מודפסת.

שימו לב שניתן להשתמש בכתיב מקוצר באופן הבא:

```
public static void Main (string[] args)
{
        Box b1 = new Box (2, 4, 6);
        Console.WriteLine (b1);
}
```

במקרה זה פעולת ההדפסה, תזמן באופן אוטומטי את הפעולה ToString(), המוגדרת לגבי העצם ותפעיל אותה, ואין צורך לכתוב במפורש:

```
Console.WriteLine (b1.ToString());
```

מחלקות מוכנות

ב-C# קיימת ספרייה סטנדרטית של מחלקות הנקראת **Base Class Library** ובקיצור **BCL**. בספרייה זו נמצאות מחלקות רבות מוכנות, מחלקות בסיס, אשר ניתן להשתמש בהן. את ממשקי המחלקות ניתן למצוא על-ידי חיפוש במאגר המידע של מיקרוסופט, הנקרא MSDN (Microsoft Developer Network).

מחלקות הבסיס הרבות שב-BCL מקובצות על פי קבוצות. בכל קבוצה מרוכזות מחלקות העוסקות באותו נושא, ולכל קבוצה יש מרחב-שם, **namespace**. על מנת להשתמש במחלקה הנמצאת תחת **namespace** מסוים, עלינו "לייבא" אותה לתכנית על-ידי שימוש במילה השמורה **using**. הפקודה **using** namespace xxx; תופיע בראש המחלקה לפני הכותרת שלה.

מספר רב של מחלקות ב- C# משתמשות במחלקות המופיעות במרחב השם System. בקבוצה זו נמצאות מחלקות המספקות שירותי שפה בסיסיים, כגון כל הטיפוסים המוגדרים מראש בשפה (**int, double** ועוד), וכן מחלקות בהן השתמשתם כבר בעבר – כגון המחלקה string – מחרוזת. כדי להפוך את כל המחלקות שבמרחב שם זה לזמינות בתוכנית כלשהי, יש לכתוב בראש קוד המחלקה, את הפקודה:

```
using namespace System;
```

וכך כל המחלקות תחת אותו מרחב שם יהיו נגישות במהלך התכנית.

<u>שימו לב</u> שניתן לוותר על ציון מפורש של המילה **namespace**.

כאשר רוצים להשתמש במחלקה Console, אשר תפקידה לכתוב מחרוזות על המסך, יש לבדוק באיזה מרחב-שם היא נמצאת. לאחר חיפוש קצר ב-MSDN, ניתן לראות שמרחב השם אליו

שייכת המחלקה הינו System. על מנת לייבא את המחלקה לתכנית שלנו, יש לכתוב בתחילת הקוד, לפני כותרת המחלקה את השורה הבאה:

using namespace System.Console;

ובקיצור אפשר לכתוב רק: **using** System;.

בהמשך תוכנית שייבאה מחלקות שונות, יש לציין בכל פעם מתוך איזו מחלקה מופעלת פעולה שבה אנחנו מעוניינים, ואי אפשר לוותר על שם המחלקה.

כך למשל חובה לציין: Console**.**Write (“hi”);

יש מחלקות שאין צורך לייבאן באופן מפורש, מכיוון שהן "מיובאות" על ידי סביבת העבודה באופן אוטומטי. תחת הגדרה זו נמצאות מחלקות שמרבים מאד להשתמש בהן (כגון System.Int32, System.Double ועוד), ולכן קצרו את הליך ההבאה שלהן.

עקב ריבוי השימוש במחלקה System.string קיצרו גם את דרך היצירה של מופעים מסוגה. זו המחלקה היחידה בה אין צורך וכמעט שאין להשתמש במילה השמורה **new** על מנת ליצור עצם.

בוודאי מוכרת לכם הצורה הבאה ליצירת המחרוזת:

```
string str = "shalom";
```

הפעולה הבונה של המחרוזת מופעלת כאשר כותבים "shalom" והעצם שנוצר מוצב בהפניה str.

** ועוד לגבי מחרוזות: ניתן בהחלט להגדיר:

```
char[] chars = { 'a', 'b' };
string s = new string(chars);
```

אך אין להגדיר:

```
string s = new string("shalom");
```

סיכום

- תהליך התכנות בשפות מונחות עצמים כולל כתיבת מחלקות המגדירות טיפוסים חדשים, יצירת עצמים מטיפוסים אלו והפעלתם באופן שיביא למילוי מטלה שהוגדרה מראש.
- עצם מורכב מתכונות המאפיינות את מצבו ופעולות שהוא יכול לבצע.
- על מנת ליצור עצם יש לזמן את הפעולה הבונה שלו בעזרת המילה השמורה **new**. יוצאת דופן היא המחלקה string שבה ניתן, רצוי ונהוג לוותר על השימוש ב-**new**.
- עצם מוחזק על ידי הפניה במשתנה מטיפוס המחלקה ממנה נוצר.
- אחרי יצירת עצם ניתן להפעיל את פעולותיו באמצעות סימון-נקודה.
- על מנת לדעת מהן הפעולות המוגדרות עבור עצם מסוים, יש לפנות לממשק של המחלקה. הממשק נוצר מתוך תיעוד המחלקה ומופיעות בו כותרות הפעולות וכן תיאור קצר של כל פעולה. הממשק אינו מספק מידע על אופן המימוש של הפעולות ובכך מיישם את עקרון ההפרדה בין ממשק למימוש.
- בסי-שרפ קיים אוסף של מחלקות מוכנות הנקרא Base Class Library. באוסף זה מאוגדות המחלקות תחת מרחבי שם, namespaces. על מנת להשתמש במחלקה יש צורך "לייבא" את מרחב השם המתאים, באמצעות המילה השמורה **using**.
- דרך נוחה להצגת מחלקות ועצמים היא בעזרת שפת המידול UML.

מושגים

"ייבא" (מרחב שם)	using
הפניה	reference
כותרת פעולה	method header
מופע	instance
מחלקה	class
ממשק	interface
סימון הנקודה	dot notation
עצם	object
פעולה	method
פעולה בונה	constructor
שפת מידול	UML
תכונה	attribute

פרק 2 דף עבודה מס' 1
קופסה צבעונית

מטרות

תרגול תיאורטי של מבנה ממשק, כותרות של פעולות והפעלתן.

מה עליכם לעשות?

לפניכם ממשק הדומה מאוד לממשק Box שהוצג בפרק עם הבדל אחד: לקופסה הזו יש תכונה נוספת שהיא: צבע.

א. השלימו את הכותרות המתאימות לתיאור הפעולות. הפעולות המפורטות הן פעולות נוספות לפעולות המופיעות בפרק, פרט לפעולה הבונה שהשתנתה:

תיאור הפעולה	כותרת הפעולה
הפעולה הבונה של המחלקה קופסה צבעונית. הפעולה מקבלת ארבע פרמטרים: אורך, רוחב, גובה וצבע ובונה קופסה חדשה בהתאם לפרמטרים אלו	ColoredBox (**double** length, **double** width, **double** height, String color)
הפעולה משנה את צבע הקופסה הנוכחית בהתאם לפרמטר	**השלימו כאן**
הפעולה מחזירה את צבע הקופסה הנוכחית	**השלימו כאן**
הפעולה מחזירה מחרוזת המתארת את הקופסה	**השלימו כאן**

ב. מדוע הפעולה הבונה השתנתה?

ג. כתבו פעולה ראשית המבצעת את המשימות הבאות:

- מייצרת שתי קופסאות שונות
- משנה לקופסה ראשונה את הצבע
- משנה לקופסה השנייה את הרוחב
- מדפיסה מחרוזות המתארות את הקופסאות לפני ואחרי השינוי.

בהצלחה!

פרק 2 דף עבודה מס' 2

דלי - Bucket

מטרות

תרגיל תיאורטי של הבנת קוד על פי ממשק ותרגול השימוש בהפניות.

ממשק המחלקה Bucket

תיאור הפעולה	כותרת הפעולה
הפעולה הבונה מקבלת כפרמטר את קיבולת הדלי ובונה דלי חדש ריק בגודל זה	Bucket (**int** capacity)
הפעולה מרוקנת את הדלי הנוכחי	**void** Empty()
הפעולה בודקת את מצב הדלי. אם הדלי הנוכחי ריק, מחזירה "אמת" ואם לא מחזירה "שקר"	**bool** IsEmpty()
הפעולה מקבלת כפרמטר כמות של מים וממלאת את הדלי הנוכחי בכמות זו. אם כמות המים היא מעבר לקיבולת הדלי, הדלי מתמלא ויתר המים נשפכים החוצה	**void** Fill (**double** amountToFill)
הפעולה מעבירה את כמות המים המקסימלית האפשרית מהדלי הנוכחי לדלי שהתקבל כפרמטר	**void** PourInto (Bucket bucketInto)
הפעולה מחזירה מחרוזת המתארת את הדלי הנוכחי בצורה הבאה: The capacity: <capacity> The current amount of water: <current amount of water>	string ToString()

מה עליכם לעשות?

1. חשבו מה צריכות להיות התכונות של המחלקה Bucket. ציירו UML של המחלקה.
2. בתרגיל זה עליכם לעקוב אחרי הקוד בפעולה הראשית ולציין מה יודפס בכל שלב:

```
public static void Main(string[] args)
{
        Bucket b1 = new Bucket(5);
        Bucket b2 = new Bucket(8);
        b1.Fill(7);
        b2.Fill(1);
        b2.Empty();
        Console.WriteLine (b1);
        Console.WriteLine (b2);
        b1.PourInto (b2);
        Console.WriteLine (b1);
        Console.WriteLine (b2);
}
```

בהצלחה!

פרק 2 דף עבודה מס׳ 3

מתוך BCL : המחלקה Point

מטרות

תרגול מעשי של הבנת הממשק ושימוש במחלקות קיימות.

המחלקה Point (מתוך BCL)

בתרגיל זה נשתמש באחת המחלקות המוכנות ב-BCL ששמה Point. המחלקה Point מייצגת נקודה דו ממדית. אנו נביא כאן ממשק חלקי שלה.
ממשק מלא קיים ונגיש דרך סביבת העבודה.

הפעולה בונה נקודה בהתאם לפרמטרים x ו-y	Point (**int** x, **int** y)
הפעולה מוסיפה לערך ה-x של הנקודה את הפרמטר dx ולערך ה-y של הנקודה את הפרמטר dy	**void** Offset (**int** dx, **int** dy)
הפעולה מחזירה מחרוזת המתארת את הנקודה	string ToString()

על מנת להשתמש במחלקה זו עלינו ״לייבא״ אותה. המחלקה Point נמצאת בחבילה הנקראת System.Drawing. זו היא חבילה המכילה את כל המחלקות הגרפיות של C#.. על מנת ״לייבא״ אותה עלינו לרשום בתחילת הקובץ (לפני הגדרת המחלקה):

```
using System.Drawing;
```

מה עליכם לעשות?

א. עליכם ליצור שתי נקודות שונות, להדפיס אותן. לאחר מכן עליכם להזיז את הנקודות למקום חדש.

ב. 1. מה לדעתכם יקרה כאשר נכתוב את שתי השורות הבאות:

```
Point p;
p.Offset (3, 4);
```

2. בדקו בעזרת הפעולה הראשית האם התוצאה היא כפי שציפיתם.

בהצלחה!

פרק 2 דף עבודה מס' 4

מתוך BCL : המחלקה string

מטרות

תרגול מעשי : הבנת ממשק ושימוש במחלקות קיימות.

* בתרגיל זה תשתמשו במחלקה מתוך BCL, המחלקה string.

המחלקה string

לפניכם ממשק חלקי של המחלקה :

bool EndsWith (string suffix)	הפעולה מקבלת מחרוזת כפרמטר. מחזירה "אמת" אם מחרוזת הפרמטר מופיעה כתת-מחרוזת בסוף המחרוזת המבצעת את הפעולה, אחרת מחזירה "שקר"
string Replace(**char** oldChar, **char** newChar)	הפעולה מקבלת כפרמטר שני תווים : oldChar ו-newChar. מחזירה מחרוזת חדשה שבה כל המופעים של oldChar מוחלפים על ידי המופעים של newChar
string ToLower()	הפעולה מחזירה מחרוזת חדשה שבה כל התווים נכתבים באותיות קטנות
string ToUpper()	הפעולה מחזירה מחרוזת חדשה שבה כל התווים נכתבים באותיות גדולות

שימו לב : המחרוזת שמפעילה את הפעולות **לעולם** אינה משתנה. כאשר נדרש שינוי היא מחזירה מחרוזת חדשה ובה השינוי. לדוגמה :

```
string str = "Moshe";
str.ToUpper();
Console.WriteLine (str);
```

ההדפסה של הקוד היא :

```
Moshe
```

כיוון שאין שינוי במחרוזת המקורית.

לעומת זאת הקוד הבא:

```
string str1= "Moshe";
string str2 = str1.ToUpper();
Console.WriteLine (str2);
```

ידפיס:

```
MOSHE
```

מה עליכם לעשות?

א. מה יודפס במהלך הפעולה הראשית הבאה:

```
public static void Main(string[] args)
{
        string s1 = "Moshe";
        string s2 = "Avraham";
        Console.WriteLine (s1[2]);
        Console.WriteLine (s2.ToUpper());
        Console.WriteLine (s2.Replace ('a', 'o'));
        Console.WriteLine (s2);
}
```

ב. כתבו תוכנית הקולטת 5 שמות. אם השם מסתיים ב"el" התוכנית תדפיס את כל השם באותיות גדולות. אם השם מסתיים ב"iam" התוכנית תדפיס את כל השם באותיות קטנות. בכל מצב אחר, התוכנית כלל לא תדפיס את השם.

בהצלחה!