פרק 3
מחלקות

בפרק הקודם ראינו כיצד ניתן להשתמש במחלקות קיימות על מנת ליצור עצמים, אך מתכנת צריך גם לדעת להגדיר בתוכניתו מחלקות לפי הצורך.
בפרק זה נלמד להגדיר מחלקה ולממשה על סמך ממשק נתון.

מהי מחלקה?
מחלקה היא הגדרה של אוסף תכונות ואוסף פעולות. לאחר שהוגדרה מחלקה היא משמשת כתבנית שניתן ליצור ממנה עצמים. לכל עצם שנוצר מהמחלקה יש את אוסף התכונות המוגדר במחלקה, והוא יכול לבצע את הפעולות המוגדרות בה. הפעולות מכירות את התכונות ויכולות להשתמש בהן. הגדרת מחלקה היא גם הגדרת טיפוס ולכן כל עצם הנוצר מהמחלקה הוא מטיפוס זה. המחלקה מכילה הגדרה של פעולה בונה המשמשת ליצירת עצמים.

לפניכם הממשק של המחלקה "דלי" –Bucket :

הפעולה בונה דלי חדש ריק בגודל הפרמטר המציין את קיבולת הדלי	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 מחזירה את הקיבולת של הדלי הנוכחי	**double** GetCapacity()
הפעולה אחזר-כמות-דלי מחזירה את כמות המים הקיימת כרגע בדלי הנוכחי	**double** GetCurrentAmount()
הפעולה שפוך-דלי 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()

מטרתנו הראשונה בפרק היא לכתוב את המחלקה Bucket על סמך הממשק הנתון. השגת המטרה תעשה על ידי ביצוע כל השלבים המתוארים בסעיפים הבאים.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

א. הכרזה על מחלקה

הגדרת מחלקה חדשה נעשית באופן הבא:

```
public class NameOfClass
{
        // כאן ייכתב גוף המחלקה
}
```

השורה המופיעה בראש ההכרזה, נקראת **כותרת המחלקה (class header)**.

המילה השמורה **public** פותחת את כותרת המחלקה. זוהי אחת מהרשאות הגישה שסישרפ מציעה למתכנת. הסבר מפורט יותר על הרשאות גישה ומטרת השימוש בהן יופיע בסוף הפרק.

המילה השמורה **class** מציינת כי זוהי הכרזה על מחלקה. אחריה יופיע שם המחלקה וסוגריים מסולסלים, שבתוכם ייכתב גוף המחלקה.

חובה לקרוא למחלקה בשמו של הטיפוס שאותו היא מגדירה, לדוגמה: **class** Box.

א.1. מוסכמות הכתיבה

בסישרפ כמו בכל שפה, ישנן **מוסכמות** (**conventions**) לגבי סגנון הכתיבה בשפה. מוסכמות אלו, הן כללים שעל כל מתכנת בשפה לציית להם.

המוסכמות המקובלות בקשר לשמות של מחלקות, הן:

- מקובל להתחיל את שם המחלקה באות גדולה, לדוגמה: **class** Bucket (הנפתח באות הגדולה B).
- קהילת יוצרי סישרפ ומשתמשיה מעודדת שימוש בשמות בעלי משמעות. כדי לאפשר קריאה נוחה של קוד (אפילו אם הוא מכיל שמות מורכבים וארוכים), ייכתב שם המחלקה כך שכל מילה חדשה תתחיל באות גדולה.

  לדוגמה: שמה של המחלקה "תולעת ורודה שלה נקודות כחולות", ייכתב כך:

class PinkWormWithBlueDots

במהלך פרק זה נתקל בעוד מוסכמות.

מדוע עלינו להיות כפופים למוסכמות אלו? התוכנית יכולה לרוץ גם אם נחרוג מהמוסכמות, אך למוסכמות חשיבות מרובה, והשימוש בהן הוא הכרחי כדי ליצור קוד מובן, קריא ונוח לשימוש.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

א.2. קובץ חדש למחלקה החדשה

את המחלקה החדשה נשמור בקובץ הנקרא: Bucket.cs.
לצורך הנוחות ושמירת הסדר אנו ממליצים להגדיר כל מחלקה בסישרפ, כבעלת הרשאת גישה פומבית, ולשמור אותה בקובץ נפרד ששמו זהה לשם המחלקה, בתוספת הסיומת '.cs'. שימו לב, קביעת שם הקובץ בצורה זו היא הרגל מומלץ.

ב. תכונות

ב.1. בחירת תכונות

כל מחלקה מכילה הצהרות של **תכונות** ו**פעולות** המאפיינות את העצמים הנוצרים ממנה. את הפעולות ראינו בממשק. תכונות הן המשתנים הפנימיים של העצם, והן מוגדרות כ**פרטיות (private)**, לשימוש במחלקה עצמה בלבד. הגדרת התכונות היא האופן בו אנו מייצגים את המחלקה. את הייצוג הזה שאינו משמעותי למשתמש במחלקה אנו מסתירים ולכן על פי רוב הוא מוגדר כפרטי.

כדי לייצג את המחלקה, יש תחילה להחליט מהן התכונות שבהן אנו מעוניינים, בהתאם לבעיה שאותה אנו מנסים לפתור. התכונות צריכות לייצג את האפיונים החשובים לנו לגבי מצב העצם. לדוגמה, עבור המחלקה "קופסה" שאת הממשק שלה ראינו בפרק הקודם, התכונות יהיו: האורך, הרוחב והגובה. עקרונית אפשר לבחור תכונות נוספות לקופסה, כמו למשל צבע או שם, אך לצרכינו התכונות הללו מיותרות. לכל תכונה שנבחרה יתכן שיופיעו בממשק פעולות הקובעות ו/או מחזירות את ערכיה.
כאשר יוצרים עצם מטיפוס המחלקה, התכונות שלו מקבלות ערכים מסוימים. אך בעזרת הפעולות השונות, ערכים אלו יכולים להשתנות במהלך תוכנית.
התכונות יהיו הדבר הראשון שאותו נגדיר בכל מחלקה, ושלב זה ייקרא: **ייצוג המחלקה**.

ב.2. הצהרה על תכונות

איך נחליט מה הן התכונות של הדלי? קודם כל לכל דלי יש קיבולת. הערך עבור התכונה "קיבולת הדלי" יתקבל כפרמטר בעת יצירת הדלי. התכונה השנייה היא כמות המים הנוכחית. בעת יצירת הדלי הוא ריק ובמהלך התוכנית ניתן למלא אותו במים ולרוקן אותו.

```
public class Bucket
{
        // תכונות (פרטיות)
        private int capacity;
        private double currentAmount;
        …
}
```


© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

הטיפוס של התכונה המייצגת את קיבולת הדלי הוא **int** כיוון שאנו מניחים כי קיבולת הדלי היא מספר שלם (אפשר היה גם להניח אחרת), ושמה הוא capacity. טיפוס התכונה המייצגת את כמות המים הנוכחית בדלי (שאינו מיוצג כמספר שלם בהכרח) הוא **double** ושמה הוא currentAmount.

המוסכמה שננהיג לגבי שמות התכונות היא המוסכמה עבור שמות משתנים בכלל: השם ייכתב באותיות קטנות, למעט תחילתה של כל מילה פנימית חדשה, שתכתב באות גדולה.

ג. הפעולות

המחלקה היא התבנית שממנה ניתן ליצור עצמים. על מנת לעשות זאת יש צורך להקצות זיכרון עבור העצם ולהכניס ערכים התחלתיים לתכונות. את התפקיד הזה מבצעת הפעולה הבונה.

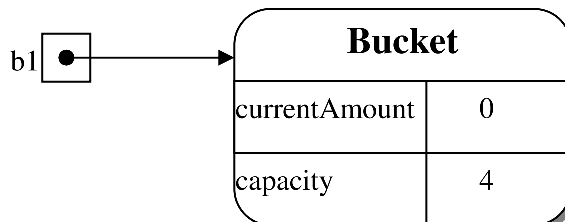
ג.1. קריאה לפעולה הבונה

לפי מה שלמדנו בפרק הקודם ניתן ליצור דליים על ידי זימון הפעולה הבונה:

Bucket b1 = **new** Bucket (4);

בעת זימון הפעולה, לפי התיאור של הממשק, נוצר דלי חדש ריק עם קיבולת 4. מה קורה בפועל בעת היצירה?

בעת הקריאה ל- **new** נוצר בזיכרון של המחשב עצם בעל שתי תכונות שהן משתנים: capacity מטיפוס **int**, ו-currentAmount מסוג **double**. אחרי יצירת העצם מתבצעת הפעולה-הבונה שתפקידה השמת הערכים בתכונות. אם לא היינו מציבים ערכים בעזרת הפעולה-הבונה היו התכונות מאותחלות לערכי ברירת המחדל של סישרפ, אך בדרך כלל אנו מעדיפים שלא להסתמך על ערכים אלו אלא לאתחל כל תכונה באופן מפורש. הפעולה מחזירה את העצם החדש והוא מוצב במשתנה מטיפוס המחלקה.



ג.2. כותרת הפעולה-הבונה

בסישרפ, אין מילה שמורה המורה על כך שפעולה מסוימת היא פעולה-בונה. לעומת זאת, יש מבנה כותרת מיוחד שבעזרתו המהדר מזהה פעולה-בונה (קונסטרקטור).

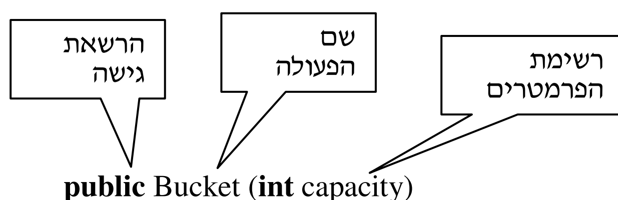
שימו לב כי המבנה המיוחד הוא הדרך היחידה לזהות קונסטרקטורים, ולכן חשוב מאוד להקפיד עליו.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

מבנה הכותרת של הפעולה-הבונה יראה כך:

1. שם הפעולה, הזהה לשם המחלקה.
2. רשימת הפרמטרים של הפעולה, בסוגריים עגולים.

ואת הכותרת מקדימה הרשאת הגישה (על פי רוב **public**).



מהו ההיגיון המצוי בבסיס מבנה זה?

- הרשאת הגישה מסוג **public** חיונית, כדי שמחלקות אחרות יוכלו להשתמש בפעולה-הבונה וליצור עצמים מטיפוס המחלקה.
- ערך החזרה של הפעולה הוא מופע חדש של המחלקה, ולכן הטיפוס של ערך החזרה הוא בדיוק כשם המחלקה. הפעולה-הבונה של המחלקה Bucket מחזירה עצם מטיפוס Bucket.
- עצם חדש נוצר על ידי זימון הפעולה-הבונה. הזהות בין שם הפעולה לשם המחלקה מאפשרת למהדר לדעת באיזו מחלקה מדובר, ולפי זה מהי תבנית העצם שצריך לבנות ומהו שטח הזיכרון הדרוש לעצם. כמו כן ידיעת המחלקה מאפשרת למהדר למצוא את הפעולה-הבונה בהכרזת המחלקה.
- שם הפעולה (הזהה לשם המחלקה) וטיפוס ערך החזרה זהים, ומאוחדים בכותרת למילה אחת.
- ולבסוף, כמו בכל פעולה, תיסגר כותרת הפעולה בסוגריים עגולים שיכילו את הפרמטרים הדרושים לביצוע הפעולה (אם יש כאלה), או יישארו ריקים (אם אין).

ג.3. גוף הפעולה-הבונה

כדי לאתחל דלי חדש, צריך לקבוע ערך התחלתי לתכונות שלו שהן קיבולת וכמות המים הנוכחית. הערך של הקיבולת יישלח על ידי מי שמבקש ליצור דלי חדש, כפרמטר של הפעולה הבונה בעת הזימון שלה. כיוון שהחלטנו כי בעת היצירה, הדלי יהיה תמיד ריק, אין צורך בפרמטר נוסף אלא ניתן לאתחל את ערך כמות המים הנוכחית ל - 0.

הפעולה-הבונה מחזירה תמיד את העצם החדש כערך החזרה של הפעולה ולכן אין להשתמש במילה **return**.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

נעיין בקוד הפעולה-הבונה של המחלקה Bucket :

```
public Bucket (int capacity)
{
        this.capacity = capacity;
        this.currentAmount = 0;
}
```

הפנייה לעצם כאשר אנחנו נמצאים בתוך קוד המחלקה, נעשית בעזרת המילה השמורה **this**. מילה זו מציינת את העצם הנוכחי, האובייקט עצמו, כלומר העצם שזה עתה נבנה.

מה שמתבצע בפעולה-הבונה שכתובה למעלה הוא השמת הערך capacity (שהתקבל כפרמטר) בתכונה capacity של העצם, והשמת 0 בתכונה currentAmount של העצם.

בעת הזימון של הפעולה-הבונה :

```
Bucket b1 = new Bucket (4);
```

הערך של הפרמטר capacity יהיה שווה 4. מעכשיו והלאה הערכים האלו שמורים בעצם ויתר הפעולות יכולות לגשת אליהם, להשתמש בהם או לשנות אותם לפי הצורך.

ג.4. פעולה-בונה ללא פרמטרים

ראינו כי הפעולה-הבונה, בדומה לכל פעולה, יכולה לקבל פרמטרים. לעתים, נרצה לכתוב פעולה-בונה שאינה מקבלת פרמטרים. כותרתה של פעולה-בונה שכזו תראה כך :

```
public Bucket()
```

פעולה כזו חייבת לאתחל את התכונות באופן כלשהו ולכן סביר שתעשה זאת על פי קבועים שאינם מועברים על ידי המשתמש :

```
public Bucket()
{
        this.capacity = 10;
        this.currentAmount = 0;
}
```

אך הפעולה יכולה גם שלא לאתחל במפורש את התכונות :

```
public Bucket()
{
}
```

במקרה זה יאותחלו התכונות לפי ערכי ברירת המחדל של סישרפ. במקרה של הדלי הן "קיבולת" והן "כמות המים הנוכחית" יאותחלו ל-0, כלומר זו תהיה פעולה-בונה שתיצור דליים "מעוותים" שגודלם 0.

כדי למנוע יצירות מעוותות שכאלה עדיף ומומלץ שלא לסמוך על ערכי ברירת המחדל של סישרפ, אלא לאתחל תמיד את הערכים של התכונות באופן מפורש.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

ג.5. פעולה-בונה ברירת מחדל

בסישרפ קיים מנגנון הדואג לכך שבכל מחלקה תופיע פעולה-בונה, גם אם המתכנת שכח או נמנע מלהוסיפה. במקרה שכזה המנגנון מוסיף למחלקה פעולה-בונה ללא פרמטרים וזו נקראת **פעולה-בונה ברירת מחדל** (**default constructor**). הפעולה הבונה תאתחל את התכונות של העצם על פי ערכי ברירת המחדל של סישרפ, מהלך שאינו מומלץ כפי שכבר ציינו לעיל.

פעולה-בונה ברירת מחדל מוספת למחלקה באופן אוטומטי אם ורק אם המתכנת לא הגדיר במחלקה אף לא פעולה-בונה אחת. את הפעולה-הבונה שסישרפ הוסיפה, אם המתכנת לא הגדיר אף פעולה-בונה, אי-אפשר לראות במפורש בקוד המחלקה, אולם היא מהווה חלק מממשק המחלקה, ומחלקות אחרות יכולות להשתמש בה כדי ליצור עצמים חדשים.

מנגנון ההוספה האוטומטית של פעולה-בונה מונע קיום מחלקות שאין אפשרות לייצר מהן עצמים. כאמור מומלץ לא לסמוך על פעולה-בונה ברירת מחדל שמוסיפה סישרפ כי לכל עצם יהיה צורך לבצע השמת ערכים בכל איברי העצם "ידנית", והדבר יוצר פתח לשגיאות הידור. לכן עלינו להוסיף לכל מחלקה פעולה בונה העונה על צורכי המחלקה.

ד. פעולות נוספות

בנוסף לפעולה-הבונה, שמטרתה ליצור עצמים והיא פעולה של המחלקה, מוגדרות במחלקה פעולות נוספות. הפעולות האלו מופעלות על ידי עצמים הנוצרים מהמחלקה ולרוב הן משתמשות בתכונותיהם. נציג ונבחן את יתר הפעולות בממשק המחלקה "דלי".

נזכיר כי הפנייה לעצמים בתוך קוד המחלקה נעשית באמצעות המילה השמורה **this** המציינת את העצם הנוכחי המבצע את הפעולות.

פעולה המרוקנת דלי:

```
public void Empty()
{
        this.currentAmount = 0;
}
```

פעולה זו מרוקנת את הדלי, כלומר משנה את כמות המים הנוכחית ל-0. מעכשיו כל פעולה שתשתמש בתכונה currentAmount תקבל את הערך 0, עד שימלאו את הדלי.

פעולה הבודקת האם דלי ריק:

```
public bool IsEmpty()
{
        return this.currentAmount == 0;
}
```

פעולה זו בודקת האם הדלי ריק, כלומר האם ערך התכונה currentAmount של הדלי הנוכחי הוא 0.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פעולה הממלאת דלי:

פעולה זו אמורה להוסיף לכמות המים הנוכחית בדלי, את הכמות שהיא מקבלת כפרמטר. אם כמות המים המתקבלת היא מעבר לקיבולת של הדלי, המים ישפכו החוצה, כלומר ילכו לאיבוד.

```
public void Fill (double amountToFill)
{
        // אם הקיבולת של הדלי קטנה מהכמות החדשה שאמורה להתקבל בדלי
        if ( this.capacity < this.currentAmount + amountToFill)
        {
                // אז מלא את הדלי עד הסוף
                this.currentAmount = this.capacity;
        }
        else
                this.currentAmount += amountToFill;
}
```

פעולה המאחזרת קיבולת:

```
public int GetCapacity()
{
        return this.capacity;
}
```

פעולה זו מחזירה את קיבולת הדלי.

פעולה המאחזרת את כמות המים:

```
public double GetCurrentAmount()
{
        return  this.currentAmount;
}
```

פעולה זו מחזירה את כמות המים הנוכחית בדלי. אם נפעיל את הפעולה על עצם שזה עתה נוצר, נקבל 0, כיוון שהדליים נוצרים ריקים. לעומת זאת אם נפעיל את הפעולה על עצם אחרי שהוא הפעיל את הפעולה Fill (**double** amountToFill) , נקבל את הכמות בדלי בזמן הקריאה לפעולה.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פעולה הממלאת דלי מתוך דלי אחר:

פעולה זו מקבלת כפרמטר עצם מטיפוס "דלי", שלתוכו היא צריכה לשפוך את המים שבדלי הנוכחי. הפעולה תמלא את דלי הפרמטר בכמות המקסימלית שאפשר להעביר מתוך הדלי הנוכחי מבלי לשפוך מים החוצה.

```
public void PourInto (Bucket bucketInto)
{
	double freeSpace = bucketInto.GetCapacity() –  bucketInto.GetCurrentAmount();
	if (this.currentAmount < freeSpace)
	{
		bucketInto.Fill (this.currentAmount);
		this.currentAmount = 0;
	}
	else
	{
		bucketInto.Fill (freeSpace);
		this.currentAmount – = freeSpace;
	}
}
```

הדלי שמתקבל כפרמטר הוא עצם קיים שנוצר לפני זימון הפעולה. כלומר בפעולה הראשית יש צורך ליצור שני דליים ולהפעיל על אחד מהם (זה שרוצים לשפוך מתוכו), את הפעולה, כאשר הדלי השני נשלח כפרמטר.

דוגמה לפעולה הראשית:

```
public static void Main (string[] args)
{
	Bucket b1 = new Bucket (5);
	Bucket b2 = new Bucket (4);
	b1.Fill (3);
	b2.Fill (4);
	b2.PourInto (b1);
}
```

יצרנו שני דליים: דלי אחד בקיבולת 5, ודלי שני בקיבולת 4. בהתאם להגדרה של הפעולה הבונה, שני הדליים נוצרו ריקים. מילאנו את הדלי הראשון בשלושה ליטרים של מים ואת הדלי השני בארבעה ליטרים של מים. בשורה האחרונה הדלי השני מנסה לשפוך את הכמות המקסימלית האפשרית של מים לתוך הדלי הראשון. כיוון שהדלי הראשון מכיל שלושה ליטרים, נותר לו מקום לשני ליטרים בלבד. לכן ישפכו רק שני ליטרים מהדלי השני לתוך הדלי הראשון. בסיום התוכנית הדלי הראשון יהיה מלא, ובדלי השני יהיו שני ליטרים של מים.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פעולה המאחזרת תיאור של דלי:

פעולה זו בונה ומחזירה מחרוזת המתארת את העצם. המחרוזת מכילה מידע שהמתכנת מעוניין להציג לגבי העצם. בדרך כלל יהיו אלו ערכי כל התכונות או חלק מהן. שימו לב שבכותרת הפעולה יש חידוש ואנו מציינים שהפעולה **מגדירה מחדש**, **override**, פעולה ישנה שנקראה באותו השם ToString(). נאמר בקיצור שהפעולה המקורית הוגדרה בעצם שהוא המקור לכל העצמים בסישרפ, העצם המוטבע מהמחלקה Object. כל המחלקות האחרות מתייחסות למחלקה זו ולכן כאשר הן מגדירות מחדש פעולה ישנה שהוגדרה ב-Object עליהן לציין שזו הגדרה מחדש בעזרת המילה override בכותרת הפעולה. נושא זה של הגדרה מחדש ילמד לעומקו במהלך היחידה החמישית.

```
public override string ToString()
{
        string str = "The capacity: " + this.capacity + "\n" +
                                "The current amount of water: " +
                                                this.currentAmount;
        return str;
}
```

הפעולה יצרה מחרוזת המכילה שרשור של ערכי התכונות של הדלי בצירוף הכותרות המתאימות. בשורה האחרונה הוחזרה המחרוזת כערך החזרה של הפעולה.

פעולה זו שימושית מאוד כאשר רוצים להדפיס את תיאור המופע:

```
public static void Main (string[] args)
{
        Bucket b1 = new Bucket (5);
        b1.Fill (3);
        Console.WriteLine (b1);
}
```

בשורה הראשונה של הפעולה הראשית נוצר דלי בקיבולת 5. בשורה השנייה מתמלא דלי זה בשלושה ליטרים של מים. בשורה האחרונה מזמן המופע b1 באופן סמוי את הפעולה ToString(). פעולה זו מחזירה את המחרוזת המתארת את המופע ואז מתבצעת ההדפסה.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

ה. הכמסה ותכנות מונחה עצמים

עקרון ההכמסה (אנקפסולציה – encapsulation) הוא מיסודותיו של תכנות מונחה עצמים. לפי עיקרון זה, עצם דומה לקופסה שחורה המספקת שירותים למשתמשים בה. מי שמשתמש בעצם, חייב להכיר את הממשק של הקופסה, כלומר את השירותים שהעצם יודע לספק. לעומת זאת, האופן בו העצם מממש את השירותים הללו צריך להיות מוחבא בתוך הקופסה השחורה, נסתר מעין המשתמש. אנקפסולציה, או הכמסה, מבטאת שאיפה 'לעטוף את המימוש בתוך קפסולה, או בכמוסה', ומכאן שמה. קל לראות כי עקרון ההכמסה עולה בקנה אחד עם עקרון הסתרת המידע, הגורס כי עצם צריך לחשוף בפני עצמים אחרים רק את שירותיו, ולהסתיר מהם את יתר הפרטים, הייצוג והמימוש, שאינם חיוניים לצורך השימוש בשירותים האלה.

בסעיף זה נכיר מנגנון חשוב המאפשר הכמסה ב-#C.

ה.1. הרשאות גישה

כאשר אנו מגדירים פעולה או תכונה ובקיצור - איבר, עלינו להחליט אילו עצמים, מופעים של אילו מחלקות, יוכלו לגשת אליו. **הרשאת גישה (access specifier)** נקבעת על פי מגדירי גישה המופיעים במחלקה כחלק מהגדרת האיבר. בסישרפ קיימות כמה רמות של הרשאות גישה, ביניהן:

1. **פומבי** (**public**) – כאשר הגדרת האיבר כוללת את מגדיר הגישה **public**, עצמים מכל המחלקות בסישרפ יכולים לגשת אליו. זוהי הרשאת הגישה שייתן המהדר כברירת מחדל, אם לא צוינה הרשאה כלשהי.
2. **פרטי** (**private**) – כאשר הגדרת האיבר כוללת את מגדיר הגישה **private**, ניתן לגשת לאיבר רק בקוד המחלקה שבה הוא מוגדר. במילים אחרות, רק המתכנת של אותה המחלקה יכול, במהלך כתיבת הקוד, להשתמש בתכונות ובפעולות הפרטיות. ובמהלך ריצת התוכנית רק עצמים מטיפוס אותה מחלקה יוכלו לפנות לתכונות ופעולות אלו.

מהאמור לעיל נובע שניתן לגשת לאיבר של עצם בשני מקרים:

- אם האיבר מוגדר כפומבי.
- אם האיבר מוגדר כפרטי והגישה אליו היא מתוך הקוד של המחלקה שהוא מוגדר בה. דוגמה מוכרת לכך היא אתחול תכונות פרטיות בתוך הפעולה-הבונה.

שימו לב, הגישה לאיבר תלויה בשני דברים: בהרשאת הגישה שלו, ובמחלקה שממנה מנסים לגשת אליו.

ה.2. הפרדה בין ממשק למימוש

באמצעות מגדירי גישה יכול כותב המחלקה להפריד בצורה קלה בין ממשק המחלקה לבין המימוש שלה. האיברים הפומביים, המוגדרים באמצעות מגדיר הגישה **public**, מרכיבים את ממשק המחלקה. עצמים ממחלקות אחרות יכולים לעבוד ולתקשר עם עצם ממחלקה זו רק דרך

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

ממשק זה, על ידי זימון פעולות פומביות או על ידי גישה לתכונות פומביות. לעומתם, האיברים הפרטיים, המוגדרים באמצעות מגדיר הגישה **private**, משמשים רק למימוש המחלקה. ניתן לזמן פעולות פרטיות או לגשת לתכונות פרטיות רק מתוך קוד המחלקה.
להפרדה הברורה בין ממשק למימוש יתרון בולט: עצמים של מחלקות אחרות אינם יכולים להשתמש באיברים הפרטיים. כל עוד הממשק של העצם נשאר זהה, אנו יכולים לשנות ולשפר את המימוש שלו מעת לעת, בלי לפגוע בתוכניות שכבר משתמשות בו.
מכיוון שכל הפעולות המופיעות בממשק הן פעולות פומביות, איננו מציינים את הרשאת הגישה **public** בטבלאות הממשק לאורך היחידה.

הדוגמה הבאה ממחישה בקצרה את הנאמר לעיל. בתוך המחלקה A ניתן לפנות ישירות לתכונה שהוגדרה כפרטית. מחוץ למחלקה A, ניתן לפנות רק אל איברים שהוגדרו כפומביים, כמו למשל אל הפעולה ()Do, אך לא ניתן לפנות ישירות לתכונה a1 שהוגדרה כפרטית:

```
public class B
{
  private A a3;

  public void Doing()
  {
     …
     Console.WriteLine ((this.a3).Do());
     Console.WriteLine ((this.a3).a1);   // לא חוקי!!!
  }
}
```

```
public class A
{
  private int a1;
    …
  public int Do()
  {
     return this.a1++;
  }

  private int DoMore (A a)
  {
     return a.a1;
  }
}
```

ה.3. פעולות עזר

מלבד הפעולות המופיעות בממשק, אלו שדרכן המחלקה מספקת את שירותיה, נמצא פעמים רבות פעולות נוספות, המסייעות לכותב המחלקה במימוש הפנימי. פעולות עזר אלו אינן צריכות להופיע בממשק המחלקה והן יוגדרו כפרטיות משום שנועדו לכותב המחלקה בלבד.

ה.4. גישה מבוקרת לתכונות

אמנם תכונות יכולות להשתנות כתוצאה מהפעלת תוכנית, אך מתכנת המחלקה יכול להחליט מראש אילו תכונות מותר לשנות ובאיזה אופן. למשל, למחלקה "דלי" יש שתי תכונות: "קיבולת" ו-"כמות המים הנוכחית". הגיוני כי התכונה הראשונה אינה ניתנת לשינוי: דלי שנוצר בקיבולת מסוימת יישאר בגודל זה. על מנת לעשות זאת ניתן להגדיר את התכונה הזאת כפרטית ולא לכתוב

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פעולה המאפשרת לשנות אותה. כך מבטיחים שקיבולת הדלי לא תוכל להשתנות על ידי המשתמשים במחלקה.
לעומת זאת התכונה "כמות המים הנוכחית" משתנה כל הזמן לאורך התוכנית, כאשר ממלאים ומרוקנים את הדלי. אך השינוי הזה בערך התכונה עצמה צריך להתבצע במגבלות של קיבולת הדלי: לא ייתכן שכמות המים בדלי תהיה מעבר לגודלו. לכן גם את התכונה הזאת כדאי להגדיר כפרטית ולספק פעולה Fill הממלאת דלי בכמות מסוימת, וכן פעולה PourInto המעבירה מים מדלי לדלי, כאשר פעולות אלו מטפלות בכל המגבלות.
בצורה זו מתכנת המחלקה יכול להבטיח שימוש תקין בעצמים של המחלקה שלו: מי שמשתמש במחלקה מוגבל לפעולות שהוגדרו עבורו, בלי יכולת גישה לתכונות עצמן. למשל, הוא אינו יכול לשנות את קיבולת הדלי, כיוון שאין לו פעולה (...) SetCapacity, ובכוונה אין לו פעולה שכזו. כמו כן המשתמש אינו יכול להכניס מים לתכונה: "כמות המים הנוכחית" מעבר לקיבולת של הדלי, כיוון שאינו יכול לגשת ישירות לתכונה, אלא רק לשנות אותה באמצעות הפעולה Fill שאינה מאפשרת הכנסת מים לדלי מעבר לקיבולת שלו.
נסיים בכלל מקובל וחשוב: על פי רוב אין לתת למשתמש גישה ישירה לתכונות של העצם. ולכן אנו מגדירים את התכונות כפרטיות.

ו. תיעוד מחלקה

עד לכאן הרבינו לדבר בשבח העבודה עם מחלקות דרך ממשקים, שהוא עיקרון מרכזי בתכנות מונחה עצמים. העבודה בעזרת ממשקים מאפשרת הפרדה מלאה בין ממשק למימוש ומייצגת את עקרון הסתרת המידע.
איך מייצרים את הממשקים המפורטים הללו? מה אמור להופיע בהם?

ו.1. כללי התיעוד

ממשק טוב אמור להכיל את מרב המידע הרלוונטי למשתמש במחלקה, מבלי לחשוף כלל את אופן המימוש שלה. הממשק צריך לספק עבור כל פעולה – תיעוד מדויק של אופן הפעילות שלה, הערכים שהיא מקבלת, ערך החזרה, מקרי גבול שיש לשים לב אליהם וערכי קצה בעייתיים שיש להימנע מלהשתמש בהם. ככל שהממשק יהיה מפורט ומדויק יותר כך השימוש במחלקה יהיה חלק ויעיל יותר (במקביל יש להיזהר מעודף פירוט המעמיס על המשתמש נתונים מיותרים).
הממשק מורכב מהתיעוד שכתב יוצר המחלקה. כל הערה פנימית שמיועדת לעיני המתכנת בלבד תופיע בקוד המחלקה לאחר // או בין הסימונים /* בראשיתה ו- */ בסופה (הערה כזו יכולה להתפרס על כמה שורות).
הערות שהן משמעותיות לממשק, כלומר: הערה בראש המחלקה המסבירה מה תכלית המחלקה, מי כתב אותה וכד', וכן הערות מקדימות לפעולות המפרטות את מהות הפעולה, הפרמטרים שהיא מקבלת, מקרי גבול וקצה והתנהגות הפעולה במקרים אלו, יופיעו בשורה או שורות המסומנות ב-/// בתחילת כל שורה.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

שורת הערה המתחילה ב-/// מסמנת למנגנון התיעוד להעתיק את ההערה הזו מגוף הקוד אל תיעוד הממשק. לכן, חשוב שכל הערה הנוגעת לממשק המחלקה (**ורק** הערה כזו) תיכתב בשורה שמתחילה ב- ///.

בתוך ההערות הללו ישולבו נתונים נוספים שהקומפיילר יתרגם אותם למידע משמעותי בדף הממשק. נתונים נוספים אלה מצוינים בשפה הנקראת XML, שלה כללים פשוטים לכתיבה:

- דרך התיאור ב-XML הינה ב"תגים" משני סוגים: תג פתיחה <tag> ותג סגירה </tag>, וביניהם מידע. כלומר, מידע חוקי ב-XML יראה כך:

```
<tag> This is a tag </tag>
```

- לעתים, יש צורך גם לספק מידע אודות "תכונה" של תג, וניתן לעשות זאת כך:

```
<tag color="red"> This is a red tag </tag>
```

כאן, קבענו שהצבע של התג אדום. שימו לב, שיש צורך במרכאות (גם למספרים), וניתן לפרט מספר בלתי מוגבל של תכונות (כלומר <tag color="red" size="big"….> הינו תג חוקי לגמרי.

- ניתן "לקנן" כמה תגים של XML אחד בתוך השני – כל עוד "סוגרים" כל תג פנימי לפני החיצוני:

```
<tag > This is a <color> red </color> tag </tag>
```

דוגמה לשימוש בתיעוד XML לממשק המחלקה Bucket:

```
/// <summary>
/// פעולה הממלאת דלי מתוך דלי אחר
/// </summary>
/// <param name="bucketInto">
/// הדלי שאליו יש למלא תכולה מהדלי הנוכחי
/// </param>
public void PourInto (Bucket bucketInto) { … }
```


© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

ו.2. סיכום ביניים – המחלקה Bucket

לפני שנמשיך, נעיף מבט במחלקת הדלי המלאה כפי שכתבנו אותה בשלבים עד כה. למחלקה זו נוסיף תיעוד מלא על פי הכללים שפירטנו לעיל.

```
/// <summary>
/// מחלקה זו מגדירה דלי
/// </summary>
public class Bucket  {

        // תכונות (פרטיות)
        private int capacity;
        private double currentAmount;

        /// <summary>
        /// הפעולה הבונה מקבלת כפרמטרים את קיבולת הדלי
        /// בונה דלי חדש ריק בגודל זה
        /// </summary>
        ///  <param name="capacity"> קיבולת הדלי  </param>
        public Bucket (int capacity)
        {
                this.capacity = capacity;
                this.currentAmount = 0;
        }

        /// <summary>
        /// הפעולה מרוקנת את הדלי הנוכחי
        /// </summary>
        public void Empty()
        {
                this.currentAmount = 0;
        }
```


© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

```
/// <summary>
/// הפעולה בודקת את מצב הדלי. אם הדלי הנוכחי ריק, מחזירה אמת,
/// ואם לא, מחזירה שקר.
/// </summary>
/// <returns>ערך בוליאני 'אמת' או 'שקר' המציין את מצב הדלי</returns>
public bool IsEmpty()
{
    return this.currentAmount == 0;
}

/// <summary>
/// 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.
/// </summary>
/// <param name="amountToFill"> כמות המים בה יש למלא את הדלי </param>
public void Fill (double amountToFill)
{
    // אם כמות המים קטנה מדי
    if (this.capacity < this.currentAmount + amountToFill)
    {
        // מלא את הדלי עד סופו
        this.currentAmount = this.capacity;
    }
    else
        this.currentAmount += amountToFill;
}

/// <summary>
/// הפעולה מחזירה את קיבולת הדלי הנוכחי
/// </summary>
/// <returns> מספר שלם – קיבולת הדלי הנוכחי </returns>
public int GetCapacity()
{
    return this.capacity;
}
```


© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

```
        /// <summary>
        ///    הפעולה מחזירה את כמות המים הקיימת כרגע בדלי הנוכחי
        /// </summary>
        /// <returns> כמות המים בדלי הנוכחי </returns>
        public double GetCurrentAmount()
        {
                return  this.currentAmount;
        }


        /// <summary>
        ///  הפעולה מעבירה את כמות המים המקסימלית האפשרית
        ///  מהדלי הנוכחי לדלי שהתקבל כפרמטר
        /// </summary>
        /// <param name="bucketInto"> דלי היעד אליו יש לשפוך את המים מהדלי הנוכחי </param>
        public void PourInto (Bucket bucketInto)
        {
                double  freeSpace = bucketInto.GetCapacity()-
                                        bucketInto.GetCurrentAmount();

                if (this.currentAmount < freeSpace)
                {
                        bucketInto.Fill (this.currentAmount);
                        this.currentAmount = 0;
                }
                else
                {
                        bucketInto.Fill (freeSpace);
                        this.currentAmount -= freeSpace;
                }
        }
}// סוף המחלקה
```


© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

ו.3. קובץ התיעוד

בהינתן מנגנון תיעוד מתאים נוכל להפוך קובץ מתועד היטב, על פי הכללים שהגדרנו, לקובץ טקסט שאינו מכיל שום מידע ורמז לקוד שבמחלקה. באופן שכזה נוכל להסתיר את המימוש מעיני המשתמש ולאפשר לו לעבוד רק דרך מסמך התיעוד שיחזיק.

נציין כי ג׳אווה מכילה מנגנון אוטומטי מהיר הנקרא javadoc המבצע את ההפרדה בין הממשק למימוש ומוציא מתוך קוד המחלקה מסמך תיעוד (API) מתאים.

ז. העמסת פעולות

ניתן לכתוב כמה פעולות בעלות אותו שם בתנאי שרשימת הפרמטרים שלהן שונה. השונות יכולה להיות במספר הפרמטרים, בטיפוסיהם או בסדר הופעת הפרמטרים בסוגריים. למשל אם היינו רוצים להגדיר פעולה ששופכת לדלי אחר לא את כל הכמות האפשרית, אלא כמות מים מוגדרת המועברת אליה כפרמטר. יכולנו להגדיר פעולה בעלת הכותרת הבאה:

public void PourInto (Bucket bucketInto, **double** amountToPour)

זאת למרות שיש כבר במחלקה פעולה בעלת שם זהה.

public void PourInto (Bucket bucketInto)

בגלל שמספר הפרמטרים או סדר הופעתם שונה, יכול המהדר לבחור את הפעולה הנכונה בעת הקריאה לפעולה. למנגנון המאפשר להגדיר מספר פעולות בעלות אותו שם והנבדלות זו מזו ברשימת הפרמטרים שלהן קוראים **העמסה** (**overloading**). שימו לב, שהשינוי חייב להיות ברשימת הפרמטרים. שינוי בטיפוס ערך החזרה אינו מספיק, כלומר אם נגדיר פעולה בשם:

public int PourInto (Bucket bucketInto)

הנבדלת מהפעולה הקיימת רק בטיפוס ערך החזרה, המהדר לא יקבל זאת. כיון שזימון הפעולה מאופיין רק על ידי שם הפעולה והפרמטרים המועברים אליה, המהדר אינו יכול להבדיל בין פעולות שונות רק על פי טיפוס ערך החזרה שכלל אינו כלול בזימון הפעולה. בקריאה לפעולה:

// יצירת שני עצמים מטיפוס Bucket בשם b1, ו-b2 //

b1.PourInto (b2);

אין למהדר מספיק מידע כדי להבדיל בין הפעולות ולדעת לאיזו פעולה קוראים: הפעולה בעלת ערך החזרה **int** או זו המחזירה **void** .

באמצעות מנגנון ההעמסה, ניתן לכתוב גם מספר פעולות-בונות שונות בהתאם לרצון המתכנת. למשל אם רוצים להגדיר פעולה-בונה שיוצרת דליים לא ריקים, אלא בעלי כמות מים התחלתית מסוימת, ניתן להגדיר פעולה-בונה בעלת הכותרת הבאה:

public Bucket (**int** capacity, **double** currentAmount)

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

כמו כן ניתן להגדיר פעולה-בונה ללא פרמטרים:

public Bucket()

כאשר מממשים פעולה זו, על המתכנת להחליט מה יהיו ערכי התכונות. תזכורת: המהדר יגדיר פעולה-בונה ברירת מחדל באופן אוטומטי רק אם במחלקה כלל לא הוגדרו פעולות בונות.

ח. איברי-מחלקה (Class Members)

בלימודיכם הקודמים כבר פגשתם את המילה **static** כאשר כתבתם פעולות שהיו סטטיות. אולי אפילו נתקלתם בתכונות שהוגדרו בעזרת **static**. כלומר, יש מצבים בהם נגדיר איברים שונים בצירוף המושג **static**. מה המשמעות של המילה הזו? מה הן פעולות ותכונות סטטיות?

איברים סטטיים הם איברים השייכים למחלקה. תחילה נלמד מהן תכונות-מחלקה, ובהמשך נערוך היכרות עם פעולות-מחלקה. (כזכור תכונות ופעולות ביחד נקראות איברים.)

כל התכונות שהכרנו עד כה קרויות תכונות-מופע, משום שהן שייכות תמיד לעצם מסוים, ומשקפות את מצבו. לעומתן, קיימות תכונות מסוג אחר, אשר אינן שייכות למופע מסוים של המחלקה, אלא למחלקה עצמה. תכונות אלו קרויות תכונות-מחלקה. הן מוקצות בזיכרון מיד כאשר מתבצעת הגדרה של מחלקה, לפני יצירתו של מופע כלשהו מטיפוס המחלקה, וניתן לגשת אליהן, לקריאה או לשינוי, על ידי פנייה למחלקה.

יתכן שזו מחלקה שכלל לא נועדה שייצרו ממנה עצמים, אלא היא מספקת שירותים לעצמים ממחלקות אחרות. במקרה שכזה התכונות שלה מאחסנות ערכים הקשורים לשירותים אלו.

כאשר המחלקה משמשת כתבנית ליצירת עצמים ונוצרו עצמים בעלי תכונות סטטיות, ניתן לגשת לתכונות אלו רק על ידי פנייה ישירה למחלקה. במקרה זה ניתן להסתכל על תכונות מחלקה כמשקפות מצב משותף של **כל** העצמים שייוצרו מאותה מחלקה, ולא כמשקפות את מצבו של עצם זה או אחר שלה. ניתן לגשת לתכונת מחלקה ואף לשנות את ערכה מתוך כל אחד מהמופעים של המחלקה.

נגדיר אם כן:

- **תכונת מופע (instance variable)**:
  1. מאפיינת מופע מסוים של המחלקה.
  2. לכל מופע יש עותק משלו של התכונה. שינוי ערך התכונה במופע אחד לא ישפיע על ערכי התכונה במופעים אחרים.

למעשה, כל התכונות שכללנו בהגדרת המחלקה עד כה היו תכונות מופע.

- **תכונת-מחלקה (class variable)**:

  תכונה שכזו תופיע בצירוף המילה השמורה **static**.

  1. מאפיינת את המחלקה באופן כללי ולא מופע מסוים שלה.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

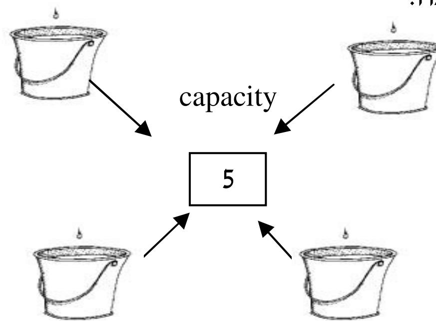
2. בזיכרון קיים עותק יחיד של תכונה זו, ללא קשר למספר המופעים שנוצרו מהמחלקה. לנקודה זו יש שתי השלכות:
   א. תכונת-מחלקה קיימת עוד לפני שנוצר המופע הראשון ואינה דורשת כלל יצירת מופעים.
   ב. אם מופע מסוים שינה את ערכה של התכונה, ערכה החדש משתקף בכל המופעים.

דוגמה לתכונת מחלקה

אם היינו רוצים שכל הדליים יהיו בעלי קיבולת אחידה, למשל - 5, ולא בגדלים שונים, היינו יכולים להגדיר את התכונה "קיבולת" של דלי כתכונת מחלקה. במקרה כזה התכונה הייתה משותפת לכל המופעים וזה היה נראה כך:

```
public class Bucket
{
        private  static int  capacity = 5;
        private  double  currentAmount;
        …
}
```

במקרה זה היה הזיכרון עבור התכונה הזו מוקצה עוד לפני יצירת המופע הראשון. לכל מופע מטיפוס "דלי" הייתה אפשרות גישה לתכונה זו. אם היינו משנים את התכונה דרך מופע אחד, הייתה קיבולת כל המופעים משתנה.



ח.1. פנייה לתכונת מחלקה

כאשר תכונת מחלקה הוגדרה כפומבית ניתן לפנות אליה רק דרך שם המחלקה:

```
Bucket.capacity
```

ולא דרך מופע מסוים של המחלקה:

```
b1.capacity            // טעות
```

כאשר התכונה פרטית אזי הפנייה אליה תעשה רק בעזרת פעולות מתאימות הקיימות בממשק, וגם אז תעשה הפנייה לתכונה דרך שם המחלקה. דרך פנייה זו המתבצעת בעזרת סימון הנקודה מדגישה את הרעיון שהתכונה היא של המחלקה עצמה.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

ח.2. פעולות-מופע ופעולות-מחלקה

כל הפעולות שעסקנו בהן עד כה היו **פעולות-מופע** (**instance methods**), כלומר פעולות שמבצע עצם מסוים (מופע) של המחלקה. **פעולות-מחלקה** (**class methods**) הן פעולות שהמחלקה מבצעת, ולא מופע מסוים שלה. פעולות כאלו ניתן לזמן גם בלי לבנות מופעים של המחלקה.

נעיר כי הנוסח 'פעולות שהמחלקה מבצעת' אינו מדויק. בפועל, **אנו** מפעילים פעולה של המחלקה כמו גם את הפעולות של העצם, אך הניסוחים הללו מתאימים לתפיסה של עצמים כישויות אקטיביות המפעילות פעולות ולכן נמשיך להשתמש בו.

פעולות מחלקה לעולם אינן משתמשות בתכונות של מופעים, אך הן יכולות להשתמש בתכונות סטטיות של המחלקה. לדוגמה, נגדיר במחלקה Bucket את הפעולה SetCapacity() המשנה את התכונה capacity של המחלקה. התכונה מוגדרת כסטטית, כלומר היא תכונה שאינה שייכת בנפרד ובאופן פרטי לשום מופע של המחלקה, לכן יש להגדיר את הפעולה כסטטית:

```
public static int SetCapacity (int capacity)
{
        Bucket.capacity = capacity;
}
```

אף את הפעולה GetCapacity() יש להגדיר כפעולה סטטית:

```
public static int GetCapacity()
{
        return Bucket.capacity;
}
```

הפעלה של פעולה סטטית אף היא נעשית דרך שם המחלקה.

ח.3. תכונות קבועות ותכונות-מחלקה

ישנן תכונות שערכן נקבע פעם אחת בלבד והוא אינו צריך להשתנות לכל אורך התוכנית. אם היינו רוצים כי קיבולת הדלי תהיה קבועה לכל הדליים שייווצרו בלי אפשרות של שינוי, היינו מגדירים את הקבוע כ-**const**. הגדרה זו כוללת בתוכה למעשה את היות הקבוע עותק יחיד, **static**, לכל המופעים:

```
public const int  CAPACITY = 5;
```

הגדרה זו קובעת שזהו קבוע (**const**), פומבי (**public**), והוא מטיפוס הנתונים הבסיסי **int**. בסישרפ מוגדר כי כל קבוע הוא גם סטטי (**static**) ולכן התכונה CAPACITY תהיה משותפת לכל הדליים שיווצרו.

התכונה מוגדרת כפומבית כיוון שלא ניתן לשנות אותה, ולכן אין סכנה לפגיעה בהכמסה. מכיוון שזהו קבוע, שמו מופיע באותיות גדולות. אגב, זוהי עוד מוסכמה של מתכנתי סישרפ.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

הגדרה זו פירושה שיתקבל עותק אחד ויחיד בלתי ניתן לשינוי של התכונה קיבולת, שתהיה בעלת ערך קבוע, שיהיה גלוי ומשותף לכל המופעים של המחלקה.

ח.4. מחלקות שירות

פעולות סטטיות נפוצות במחלקות שירות כגון המחלקה Math או המחלקה Console. מטרתן העיקרית של מחלקות אלו אינה לשמש ליצירת עצמים. הן מכילות אוסף של פעולות בנושא מסוים, כולן פעולות מחלקה.

כזו היא מחלקת Console בה השתמשנו עד כה לצורכי הדפסה על הצג וקליטת נתונים. איננו יוצרים מופעים של המחלקה. אנו משתמשים בפעולותיה כדי להדפיס נתונים תוך זימון הפעולות ישירות מתוך המחלקה, וכן לצורך קליטת נתונים מהמשתמש.

מחלקה נוספת שימושית מאד היא המחלקה Math המכילה פעולות לביצוע חישובים מתמטיים שונים. אגב, Math מיובאת לשימוש כל תוכנית באופן מיידי עם כתיבת השורה:

```
using System;
```

בראש כל תוכנית.

למשל פעולת השורש: sqrt של המחלקה Math. פעולה זו מקבלת מספר ומבצעת עליו את האלגוריתם של הוצאת שורש. כיוון שהיא לא משתמשת בתכונות מופע של Math היא פעולה סטטית. במחלקה Math גם לא מוגדרות תכונות מופע אך מוגדרות בה תכונות סטטיות: PI, ו-E המייצגות את היחס בין היקף מעגל לקוטרו, ואת בסיס הלוגריתמים הטבעיים שהם מספרים אי-רציונליים שימושיים.

לאחר הייבוא של System ניתן לזמן את הפעולה להוצאת שורש באופן הבא:

```
Console.WriteLine (Math.Sqrt(9));
```

ח.5. הפעולה הראשית Main

פעולה סטטית שימושית ביותר המוכרת לנו היא הפעולה הראשית Main. במבנה התוכניות שהכרנו עד כה, תוכנית מכילה מחלקה אחת שבה רק פעולה סטטית יחידה Main, ומחלקות נוספות שמהן מייצרים עצמים. הפעולה הראשית מופעלת מיד בתחילת הריצה של התוכנית והפקודות שבה הן היוצרות עצמים ממחלקות אחרות ושולחות אליהם הודעות לביצוע פעולות. מכיוון שהפעולה הראשית היא הפעולה הראשונה שמבוצעת על ידי המערכת, עוד לפני יצירת מופע כלשהו, היא חייבת להיות מוגדרת כפעולת מחלקה.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

ח.6. פעולות להרחבת הממשק

לעתים נרצה לבצע פעולה שאינה מוגדרת בממשק מחלקה, על מופע של מחלקה זו. למשל, על מופעי דלי נרצה לבצע פעולה המעבירה מים מדלי אחד לשני, כך שבסופה שני הדליים יהיו מלאים בכמות מים שהיא הממוצע בין שני הדליים. כותרתה של פעולה שכזו צריכה הייתה להראות כך :

```
public void PourAverage (Bucket b1)
```

כאשר יש לנו אפשרות לפתוח את הקוד של המחלקה ולשנות אותה, אפשר להגדיר את הפעולה החדשה בתוך המחלקה, אך לעיתים נרצה לבצע הוספה שכזו לגבי מחלקה שלא אנחנו הגדרנו ומימשנו. לגבי מחלקה סגורה ומקומפלת שאין לנו אפילו את קוד המקור שלה, לא נוכל לבצע את תהליך ההוספה של הפעולה לממשק המחלקה. ביחידה חמישית נלמד על מנגנון הירושה המאפשר הוספה של פעולה למחלקה שכזו אך בינתיים נוכל להיעזר ברעיון של **פעולת מחלקה**.

אנו נגדיר את הפעולה הרצויה לנו מחוץ למחלקה דלי, כפעולת מחלקה המקבלת שני דליים (כי אין מופע של דלי שמפעיל את הפעולה). לכן הכותרת שלה תכיל את הגדרתה כפעולת מחלקה ותראה כך :

```
public static void PourAverage (Bucket b1, Bucket b2)
{
        double averageAmount = (b1.GetCurrentAmount() + b2.GetCurrentAmount())/2;
        b1.Empty();
        b2.Empty();
        b1.Fill (averageAmout);
        b2.Fill (averageAmout);
}
```

אם נניח שהפעולה הוגדרה במחלקה BucketUtils אזי כמו בכל פעולה סטטית, שימוש בפעולה ייעשה בעזרת שם המחלקה :

```
BucketUtils.PourAverageAmount();
```

שימו לב שהפעולה אינה מתבצעת במחשב כפי שהיינו מבצעים אותה במציאות. במשחק דליים מציאותי היינו צריכים להיעזר בדלי עזר נוסף כדי להעביר כמויות מדלי למשנהו עד שנגיע לכמות הרצויה בכל דלי. בתכנות, מכיוון שממילא איננו משתמשים במים "מסוימים" אלא מחשבים כמויות תיאורטיות לחלוטין, אנו "נפטרים" מכל המים על ידי ריקון הדליים ואז ממלאים אותם מחדש בדיוק בכמות הרצויה.
הפעולות הסטטיות יכולות להיות מוגדרות בכל מחלקה שהיא. לעתים הן יהיו מוגדרות בקובץ שבו נמצאת הפעולה הראשית שגם היא כזכור סטטית, לפני או אחרי הפעולה הראשית עצמה.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

ט. סיכום

- ניתן לראות את המחלקה כמגדירה טיפוס נתונים חדש וכתבנית ממנה ניתן ליצור מופעים מטיפוס זה.
- לכל מחלקה יש תכונות: אלו משתנים פנימיים של כל עצם הנוצר ממנה. משתנים אלו קובעים את הייצוג של המחלקה. המשתנים מאותחלים בזמן יצירת העצם והערכים שבהם יכולים להשתנות במהלך התוכנית. ערכי המשתנים מייצגים את מצבו של העצם ברגע נתון. לרוב מוגדרות התכונות כפרטיות.
- הפעולה הבונה היא הדרך לייצר עצמים (מופעים) מן המחלקה. במימוש הפעולה-הבונה תכונות העצם מאותחלות בהתאם לפרמטרים המועברים לתכונה הבונה ובהתאם לשיקול הדעת של המתכנת.
- יתר הפעולות המוגדרות במחלקה הן פעולות של העצמים. בעזרתן ניתן לגשת לתכונות, להשתמש בערכיהן ולשנות אותם.
- הרשאות הגישה מאפשרות לקבוע אילו איברים של העצמים יהיו נגישים ואלו לא. הרשאות הגישה מאפשרות הפרדה בין הממשק למימוש, ותומכות ברעיון ההכמסה.
- אל פעולות פומביות, כולל הפעולה הראשית (...)Main, ניתן לפנות מתוך מחלקות אחרות, בניגוד לפעולות פרטיות אליהן ניתן לפנות רק בתוך קוד המחלקה עצמה.
- מנגנון התיעוד מאפשר ליישם את עקרונות ההכמסה (אנקפסולציה) ועקרונות העבודה עם ממשקים, שהם מעמודי התווך של תכנות מונחה עצמים. בעזרת מנגנון זה ניתן לייצר תיעוד לממשקים בעבור מתכנתים נוספים המשתמשים במחלקות שכתבתם, מתוך קוד המחלקה המתועד בהתאם.
- מנגנון ההעמסה מאפשר להגדיר באותה מחלקה פעולות בעלות אותו שם אך בעלות רשימת פרמטרים שונה.
- תכונות מחלקה הן תכונות השייכות למחלקה ולא למופע מסוים שלה. תכונות אלו נגישות רק ישירות מתוך המחלקה.
- פעולות מחלקה הן פעולות המבוצעות על ידי המחלקה ואינן משתמשות בתכונות מופע. דוגמה לכך הן פעולות שירות של המחלקה Math או הפעולות של המחלקה Console.
- פעולות מחלקה כולל הפעולה הראשית, ניתנות להפעלה גם קודם ליצירת מופע כלשהו.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

מושגים

class members	איברי מחלקה
this	הנוכחי
overloading	העמסה
access specifiers	הרשאות גישה
class header	כותרת מחלקה
static	סטטי (=של המחלקה)
public	פומבי
default constructor	פעולה-בונה ברירת מחדל
private	פרטי

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פרק 3 דף עבודה מס׳ 1
נקודת התחלה

מטרות

1. כתיבת מחלקה פשוטה.
2. בדיקת המחלקה בעזרת תוכנית בדיקה.

חלק א:

המחלקה Point

בפרק הקודם השתמשנו במחלקה Point הקיימת ב- BCL, ועתה נלמד לכתוב מחלקה שדומה לה.

המחלקה Point שנבנה, מגדירה נקודה דו-ממדית בעלת שתי קואורדינטות: x ו-y.

ממשק המחלקה Point

הפעולה בונה נקודה חדשה על פי ערכי הפרמטרים	Point (**double** x, **double** y)
הפעולה מחזירה את קואורדינטת ה-x של הנקודה	**double** GetX()
הפעולה מקבלת ערך מטיפוס **double**, וקובעת את קואורדינטת ה-x של הנקודה בהתאם	**void** SetX (**double** x)
הפעולה מחזירה את קואורדינטת ה-y של הנקודה	**double** GetY()
הפעולה מקבלת ערך מטיפוס **double**, וקובעת את קואורדינטת ה-y של הנקודה בהתאם	**void** SetY (**double** y)
הפעולה מחזירה מחרוזת המתארת את נתוני הנקודה על פי הצורה הבאה: (<X> , <Y>)	string ToString()

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

מה עליכם לעשות?

א. צרו מחלקה חדשה בשם Point. (זכרו: התאימו בין שם המחלקה ושם הקובץ שבו היא נמצאת).

ב. ייצגו את המחלקה Point (במילים אחרות: קבעו מה הן התכונות של המחלקה).

ג. ציירו UML המתאים למחלקה.

ד. ממשו את הפעולות שבמחלקה Point.

כדי לבדוק שהמחלקה שכתבתם עובדת כראוי, עליכם לכתוב תוכנית בדיקה, לפי ההנחיות הבאות:

המחלקה TestPoint

צרו מחלקה נוספת בשם TestPoint (כמובן שבקובץ TestPoint.cs).

בתוך המחלקה TestPoint כתבו פעולת (...)Main המבצעת את משימות הבאות:

1. בונה נקודה חדשה עם קואודינטות (43, 7).
2. בונה נקודה חדשה נוספת עם קואורדינטות (5, 5).
3. מדפיסה את שתי הנקודות בעזרת המחרוזת המוחזרת מהפעולה ()ToString, באופן הבא:

```
Console.WriteLine (xxx);
```

4. מחליפה בין קואורדינטות ה-x של שתי הנקודות, תוך שימוש בפעולות השונות של המחלקה נקודה.
5. מדפיסה שוב את הנקודות החדשות .

חלק ב:

נרחיב את המחלקה Point ונוסיף לה שתי פעולות:

הפעולה מקבלת נקודה ומחזירה את המרחק שבין הנקודה הזו לנקודה הנוכחית. (ראו למטה תזכורת לחישוב המרחק)	**double** Distance (Point p)
הפעולה מקבלת נקודה ומחזירה את הנקודה הנמצאת באמצע שבין הנקודה שהתקבלה כפרמטר ובין הנקודה הנוכחית	Point Middle (Point p)

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

תזכורת: חישוב נקודת האמצע בין שתי הנקודות (x_1, y_1) ו- (x_2, y_2) הוא

$$x_{middle} = \frac{x_1 + x_2}{2} \quad y_{middle} = \frac{y_1 + y_2}{2}$$.

חישוב המרחק בין שתי נקודות (x_1, y_1) ו- (x_2, y_2) הוא $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

פעולות החזקה והשורש קיימות במחלקה Math.

ממשק חלקי של המחלקה Math :

הפעולה מקבלת שני פרמטרים x ו-y, ומחזירה את הערך של x בחזקת y	**double** Pow (**double** x, **double** y)
הפעולה מקבלת את הפרמטר x ומחזירה את השורש הריבועי שלו	**double** Sqrt (**double** x)

הפעולות האלו הן פעולות מחלקה. הפעלת הפעולות נעשית דרך שם המחלקה. למשל על מנת להוציא שורש מ-9 יש לכתוב :

```
double root = Math.Sqrt (9);
```

1. הוסיפו לפעולה הראשית חישוב של המרחק בין שתי הנקודות המקוריות שיצרתם בחלק א, והדפסתו.
2. הוסיפו לפעולה הראשית חישוב של נקודת האמצע בין הנקודות שיצרתם בחלק א, לאחר החלפת ערכי ה-x. הדפיסו את הנקודה.

שימו לב:

בפעולה המחשבת נקודת אמצע, ערך החזרה גם הוא עצם מסוג Point. כלומר, עליכם ליצור את העצם החדש בתוך מימוש הפעולה ואז להחזיר אותו כערך החזרה של הפעולה.

בהצלחה!

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פרק 3 דף עבודה מס' 2
סימני התיעוד

מטרות

1. הכרת התגיות המשמשות לתיעוד קבצי סישרפ.

רקע

בדף עבודה זה נכיר את תגיות התיעוד המקובלות בסישרפ. בהינתן מנגנון מתאים יופק מקובץ cs. מתועד, מסמך המכיל את הממשק בלבד. כפי שצוין בפרק כל הערה המתחילה בתגית, בתוך סוגריים <xx>, תסתיים בסוגריים ובהן התגית המסיימת: </ xx>.

רשימת התגיות המקובלות:

תיאור של מחלקה / פעולה וכו' <summary>

כל מידע נוסף שאינו קשור לתיאור המחלקה / הפעולה – דוגמאות וכו' <remarks>

תיאור של פרמטר לפעולה <param>

תיאור של שגיאה, כאשר התכונה מציינת שגיאה קיימת שעשויה לקרות כתוצאה מהפעולה הנוכחית: <exception cref=”exception”>

תיאור ערך החזרה של מחלקה <returns>

תיאור איבר מחלקה <value>

למשל:

```
public class Employee
{
   private string name;
   /// <value> Name accesses the value of the name data member </value>
   public string Name
   {
      get
      {
         return name;
      }
      set
      {
         name = value;
      }
   }
}
```


© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

רשימה מלאה של תגים "מומלצים" מופיעה ב-

http://msdn.microsoft.com/library/default.asp?url=/library/enus/csref/html/vclrftagsfordocumentationcomments.asp

מה עליכם לעשות?

עליכם לתעד את הקוד של המחלקה Point*, שכתבתם בדף העבודה הקודם באופן מלא:

שימו לב להוסיף היכן שצריך: returns אם הפעולה אינה **void**,

param אם יש פרמטרים, וכו'.

* ביכולתכם לתעד כל מחלקה אחרת שכתבתם אך השתדלו שזו תכיל מגוון פעולות ולא רק פעולת Main, כך שתוכלו להתרשם מהממשק המתקבל.

בהצלחה!

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פרק 3 דף עבודה מס׳ 3

משחק בקוביות

מטרות

1. כתיבת UML של מחלקה.
2. כתיבת מחלקה פשוטה ושימוש בה.
3. עבודה עם ממשק.
4. תיעוד מחלקה.

המחלקה קובייה – Die

המחלקה **Die** (קובייה) מגדירה קובייה שלה 6 פאות. על הפאות מופיעים המספרים 1 עד 6. כאשר הקובייה נמצאת במנוחה, ונשאלת השאלה "מהו המספר שהקובייה מראה?" התשובה לכך היא: "המספר שנמצא על הפאה העליונה".

ממשק המחלקה Die

הפעולה בונה עצם מטיפוס Die. הקובייה שנוצרה מראה מספר אקראי בין 1-6	Die()
הפעולה מדמה "הטלת קובייה". בתום הפעולה מתעדכן המספר שהקובייה מראה לאחר ההטלה	**void** Roll()
הפעולה מחזירה את המספר שמראה הקובייה	**int** GetNum()

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

מה עליכם לעשות?

1. חשבו מהן התכונות הנחוצות למחלקה Die וציירו UML מתאים למחלקה.
2. כתבו את המחלקה **Die** במלואה (כולל תיעוד ויצירת קובץ תיעוד).

רמז: כדי להטיל את הקוביות באופן אקראי השתמשו במחלקה Random. על אופן פעולתה ראו ב- MSDN:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemrandommemberstopic.asp

3. כתבו מחלקה בשם **DiceGame** (משחק קוביות), ובה פעולה ראשית היוצרת שתי קוביות.

בכל תור תטיל התוכנית את שתי הקוביות עד אשר יתקבל הצירוף: 6, 6. בכל תור יש למעשה שתי הטלות של שתי קוביות המשחק.

2. התוכנית תדפיס את תוצאות ההטלות בכל התורות.
3. כאשר יתקבל הצירוף 6, 6 תיעצר התוכנית ותדפיס כמה תורות התקיימו עד אשר קיבלנו 6, 6.
4. תעדו את המחלקה במלואה.

בהצלחה!

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פרק 3 דף עבודה מס' 4

תאריך (Date)

מטרות

5. כתיבת מחלקה פשוטה ותיעודה.
6. תרגול תיאורטי של הנושאים הנלמדים בפרק.
7. עבודה עם ממשק.
8. מעקב אחר קוד.

המחלקה Date

המחלקה Date מגדירה את הטיפוס תאריך המורכב מיום, מחודש ומשנה.
הממשק הבא מגדיר אילו פעולות פומביות יש למחלקה. התכונות (הפרטיות) אינן מוזכרות כלל, ועל המתכנת עצמו להחליט אילו תכונות הוא רוצה וצריך שיהיו למחלקה.

הפעולה בונה עצם מטיפוס Date על פי פרמטרים נתונים	Date (**int** day, **int** month, **int** year)
הפעולה מחזירה את השנה	**int** GetYear()
הפעולה מחזירה את החודש	**int** GetMonth()
הפעולה מחזירה את היום	**int** GetDay()
הפעולה קובעת את ערך השנה על פי הפרמטר הנתון. ערך הפרמטר הוא מספר שלם לא-שלילי בן ארבע ספרות	**void** SetYear (**int** yearToSet)
הפעולה קובעת את ערך החודש על פי הפרמטר הנתון. ערך הפרמטר הוא מספר שלם בין 1 ל-12	**void** SetMonth (**int** monthToSet)
הפעולה קובעת את ערך היום על פי הפרמטר הנתון. ערך הפרמטר הוא מספר שלם בין 1 ל-31	**void** SetDay (**int** dayToSet)
הפעולה מחזירה **true** אם ורק אם התאריך קודם לתאריך שהועבר אל הפעולה כפרמטר	**bool** Before (Date other)
הפעולה מחזירה מחרוזת המתארת את התאריך בצורה הבאה: <day>/<month>/<year>	string ToString()

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

מה עליכם לעשות?

חלק א:

1. כתבו את כותרת המחלקה Date, ייצגו את המחלקה וממשו את כל הפעולות הנזכרות בממשק. ניתן להניח תקינות של כל הקלטים ואין צורך לבצע בדיקות תקינות לגביהם.
2. תעדו את המחלקה כראוי.
3. כתבו תוכנית בדיקה בשם TestDate, ובה בדקו את כל הפעולות שמימשתם במחלקה Date, כלומר צרו לפחות שני מופעים של Date, שיפעילו את כל פעולות הממשק.

חלק ב:

לפניכם תוכנית ראשית המשתמשת במחלקה Date:

```
public static void Main (string[] args)
{
    Date d1 = new Date (16, 7, 1963);
    Date d2 = d1;
    d1.SetDay (20);
    d2.SetYear (1980);
    Console.WriteLine (d1);
    Console.WriteLine (d2);
}
```

1. מה יודפס בתום הרצת התוכנית?
2. כמה עצמים מסוג Date נוצרו במחלקה הראשית? הסבירו.

בהצלחה!

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פרק 3 דף עבודה מס׳ 5
מספר רציונלי

מטרות

1. כתיבת מחלקה ותיעודה.
2. עבודה עם ממשק.
3. הגדרה ומימוש של פעולה חיצונית (**static**).

רקע

מספר רציונלי הוא מספר הניתן לכתיבה כמנה של שני מספרים שלמים: מונה ומכנה. למשל, 0.3 הוא מספר רציונלי כיוון שהוא ניתן לכתיבה כ- $\frac{3}{10}$.

טיפוס כזה כבר קיים בסישרפ כטיפוס פשוט (**double**). עתה נגדיר אותו בתור מחלקה.

ניתן להגדיר מספר רציונלי בעזרת מחלקה בעלת שתי תכונות: מונה ומכנה, כאשר שניהם מספרים שלמים.
הכפלת המונה והמכנה של מספר רציונלי באותו מספר, מבטאת ייצוג אחר של אותו מספר.

לדוגמה: $\frac{1}{3}=\frac{2}{6}=\frac{3}{9}$

כלומר: יכולים להתקיים עצמים המייצגים אותו מספר רציונלי למרות שערכי תכונותיהם שונים.
לא כל שני מספרים מייצגים מספר רציונלי חוקי. כאשר ערך התכונה המייצגת את המכנה הוא 0, המספר איננו חוקי.

המחלקה Rational

לפניכם ממשק המחלקה Rational המגדירה מספר רציונלי.

תזכורת: בפעולות רבות יש להתחשב במקרי קצה בעייתיים. כאשר מדובר על קלט לפעולה נעדיף להתריע בתיעוד הפעולה על הבעיה ולקבוע עבור אילו ערכים תפעל הפעולה כראוי. כך נמנע את המשתמש במחלקה להעביר ערכים לא רצויים לפעולה. בהמשך לימודיכם תלמדו על מנגנון החריגות שמאפשר להתמודד לעומק עם מקרים אלו ולהציע להם פתרונות. בבעיות העלולות להתעורר תוך כדי הפעולה נצטרך לבדוק ולטפל תוך כדי מימוש הבעיה.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

הפעולה הבונה מקבלת שני פרמטרים : x עבור המונה ו-y עבור המכנה. הנחה : המכנה לא יכול להיות שווה 0, כלומר כל מספר שנוצר הוא מספר רציונלי חוקי	Rational (**int** x, **int** y)
הפעולה מחזירה את המונה של המספר הנוכחי	**int** GetNum ()
הפעולה מחזירה את המכנה של המספר הנוכחי	**int** GetDenom-()
הפעולה מקבלת כפרמטר מספר רציונלי נוסף num, ובודקת האם שני המספרים הרציונליים שווים זה לזה	**bool** IsEqual (Rational num)*
הפעולה מקבלת כפרמטר מספר רציונלי נוסף num, ומחזירה רציונלי שהוא מכפלת הפרמטר במספר הנוכחי	Rational Multiply (Rational num)
הפעולה מקבלת כפרמטר מספר רציונלי נוסף num, ומחזירה מספר רציונלי שהוא המנה של המספר הנוכחי ב-num. יש לבדוק שהמחלק אינו בעל **מונה** שווה ל-0, אם המחלק אכן לא תקין תחזיר הפעולה **null**	Rational Divide (Rational num)**
הפעולה מחזירה מחרוזת המתארת את המספר הרציונלי בצורה הבאה : <y> / <x>	string ToString()

* השוואה של מספרים רציונליים תיעשה בעזרת מכפלת המונה של האחד במכנה של השני והשוואת המכפלות. לדוגמה:

$$\frac{a}{b} \stackrel{?}{=} \frac{c}{d} \qquad a \cdot d \stackrel{?}{=} c \cdot b$$

** החלוקה של שבר בשבר נעשית על ידי כפל השבר הראשון בהופכי של השבר השני. לדוגמה :

$$\frac{2}{3} \div \frac{4}{6} = \frac{2}{3} \cdot \frac{6}{4}$$

*** בדברנו על מספרים רציונליים איננו מתעסקים עם החלקים השבורים המתקבלים כתוצאה של פעולות חילוק, ולא נהיה מעוניינים לקבל מידע לגבי חלקים לא שלמים אלו.

מה עליכם לעשות?

חלק א

1. כתבו את המחלקה Rational. הקפידו לתעד כראוי את הפעולה הבונה תוך ציון הדרישה שלא להכניס 0 במכנה.
2. כתבו תוכנית בדיקה הבודקת את המימוש שכתבתם עבור פעולות הממשק.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

חלק ב

קבלתם את המחלקה Rational כמחלקה קיימת, ואינכם יכולים לשנותה. אתם מעוניינים בשתי פעולות נוספות: פעולת חיבור ופעולת חיסור של שני מספרים רציונליים.

1. היכן ואיך תוכלו להגדיר ולממש פעולות אלו?
2. כתבו את כותרות הפעולות המתאימות וכן תיעוד מלא לפעולות אלה.
3. ממשו את פעולת החיבור של שני מספרים רציונליים.
4. כתבו תוכנית בדיקה שתוכיח את נכונות הפעולות שמימשתם.

בהצלחה!

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פרק 3 דף עבודה מס' 6

הרשאות גישה

מטרות

תרגול תיאורטי בהרשאות גישה.

מה עליכם לעשות?

בטבלה שלפניכם מופיע קוד חלקי של שתי מחלקות: Alice ו-Bob.

```
public class Bob
{
  private Alice ally;

  public Bob()
  {
    this.ally = new Alice();
  }

  public void ChangeAlice()
  {
    Console.WriteLine ((this.ally).a1);
    (this.ally).IncreaseBoth();

     Console.WriteLine ((this.ally).a2);
    (this.ally).FindA2 (this.ally);
  }

}
```

```
public class Alice
{
  public int a1;
  private int a2;
  public Alice()
  {
    this.a1 = 1;
    this.a2 = 2;
  }
    public void IncreaseBoth()
   {
      this.a1++;
      this.a2++;
   }
   private int FindA2 (Alice a)
   {
     return a.a2;
   }
}
```

ענו על השאלות הבאות:

1. האם המחלקה Alice, על פי הקטע המופיע לעיל, תעבור הידור? אם לא, נמקו והדגימו.
2. המחלקה Bob אינה עוברת הידור (קומפילציה). נבדוק מדוע:
   i. האם הפנייה לתכונה a1 בשורה הבאה היא פנייה חוקית או לא? הסבירו:
   Console.WriteLine ((**this**.ally).a1);
   ii. האם בשורה (**this**.ally).IncreaseBoth(); מותר לזמן פעולה של המחלקה Alice?
   iii. האם השורה: Console.WriteLine ((**this**.ally).a2); תקינה? הסבירו ונמקו.
   iv. האם מותר לזמן בתוך מחלקה Bob את הפעולה:
   (**this**.ally).FindA2 (**this**.ally), הסבירו.

בהצלחה!

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פרק 3 דף עבודה מסי 7

מונה – counter

מטרות

תרגול השימוש בתכונה סטטית.

רקע

אנו מעוניינים שכל אחת מהנקודות הנוצרות על ידי המחלקה Point תקבל מספר סידורי המציין איזה מופע היא של המחלקה.

כלומר, הנקודה הראשונה שתיווצר מהמחלקה תקבל את המספר הסידורי: 1, הנקודה שאחריה תקבל את המספר: 2. הנקודה ה-n תקבל את המספר הסידורי: n.

מה עליכם לעשות?

1. הוסיפו למחלקה Point את התכונה **או** התכונות הנדרשות לצורך מתן מספר סידורי לכל מופע של המחלקה. הסבירו מה עשיתם.

   רמז: בעוד שהמספר הסידורי של כל נקודה הוא תכונה המאפיינת את העצם עצמו, מי ישמור את המספר הכולל של הנקודות שכבר נוצרו מן המחלקה?

2. שנו את מימוש הפעולה הבונה כך שכל מופע של נקודה יכיל את מספרו הסידורי מרגע יצירתו.

3. שנו את הפעולה ()ToString כך שבראשית המחרוזת המתארת את הנקודה יודפס מספרה הסידורי ורק בהמשך יופיעו שאר מאפייני הנקודה.

4. הוסיפו למחלקה פעולה המאפשרת לקבל ברגע נתון את מספר המופעים שנוצרו מטיפוס המחלקה עד כה.

5. השלימו את תיעוד המחלקה כך שיתאים לשינויים שביצעתם.

בהצלחה!

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פרק 3 דף עבודה מס' 8
פעולת מחלקה

מטרות

תרגול השימוש בפעולה סטטית המרחיבה את המחלקה.

מה עליכם לעשות ?

ברצוננו להגדיר פעולה השופכת כמות **נתונה** של מים מדלי אחד לדלי אחר, להבדיל מהפעולה שבממשק הדלי השופכת את הכמות המקסימלית האפשרית מדלי למשנהו.
איננו רוצים להוסיף פעולה זו לממשק המחלקה דלי (למרות שזו מחלקה שאנו הגדרנו ולכן זה בהחלט אפשרי), אלא לכתוב אותה כפעולה סטטית.

ענו על הסעיפים הבאים:

1. **כותרת הפעולה**: אילו פרמטרים תקבל הפעולה ומאיזה טיפוס?
2. **מיקום הפעולה**: היכן תוגדר הפעולה? אם אינכם בטוחים חזרו וקראו את הסעיפים המתאימים בפרק.
3. **מימוש הפעולה**: האם במימוש הפעולה ניתן לפנות לתכונות הדלי ישירות? כיצד תממשו את הפעולה, האם תשתמשו בפעולות ממשק כלשהן? אילו?
4. **מקרי קצה**: הגדירו מקרי קצה שיש להתחשב בהם ולתת להם מענה או לפחות לציין בתיעוד מה תעשה הפעולה במקרים אלו.
5. **מימוש והפעלה**: כתבו את הפעולה במחלקה המכילה את הפעולה הראשית ותעדו אותה. הפעילו את הפעולה כדי להוכיח את נכונות המימוש.

בהצלחה!

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

פרק 3 דף עבודה מסי 9
כיתה מוזיקלית

מטרות

תרגול השימוש בתכונה סטטית.

מה עליכם לעשות?

בבית ספר קיימת כיתה שתלמידיה אוהבים מוזיקה. לתלמידים יש ספרייה מוזיקלית משותפת. הם אוספים דיסקים ומשמיעים אותם בהפסקות. בתחילת השנה, כל תלמיד מתחייב להביא מספר מסוים של דיסקים שונים (INITIAL_AMOUNTS_OF_DISCS), מכאן ואילך הוא יכול להוסיף דיסקים אם רצונו בכך (בסוף השנה הילד שהביא הכי הרבה יקבל פרס מיוחד מבני כיתתו).
רן, חובב המוזיקה והמחשבים, רוצה לערוך מעקב אחר כמות הדיסקים שמביאים התלמידים. כדי לעשות זאת כתב מחלקה המגדירה תלמיד בכיתה.
לפניכם מחלקה שכל מופע שלה מייצג תלמיד בכיתה זו :

```
public class Student
{
        private string name;
        private int myAmount;
        private static int classDiscBox = 0;
        public const int INITIAL_AMOUNTS_OF_DISCS = 3;

        public Student (string name)
        {
                this.name = name;
                this.myAmount = INITIAL_AMOUNTS_OF_DISCS;
                Student.classDiscBox += INITIAL_AMOUNTS_OF_DISCS;
        }
        public static int GetClassDiscBox()
        {
                return Student.classDiscBox;
        }
        public int GetStudentAmount()
        {
                return this.myAmount;
        }
```


© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

```
        public string GetName()
        {
            return this.name;
        }
        public void EnterDiscs (int amount)
        {
                Student.classDiscBox += amount;
                this.myAmount += amount;
        }

}
```

1. במחלקה אחרת נכתבה הפעולה הראשית. מה תהיה תוצאת ההדפסה?

```
public class Test {
    public static void Main (string[] args)
{
        Student[] members = new Student[3];
        members[0] = new Student ("Moshe");
        members[1] = new Student ("Dvir");
        members[2] = new Student ("Michal");

       for (int i = 0; i < members.length ; i++)
       {
          members[i].EnterDiscs (i);
       }
       Console.WriteLine (members[2].GetStudentAmount());
       Console.WriteLine (Student.GetClassDiscBox());
   }
}
```

2. מדוע הוגדר המשתנה classDiscBox כ - **static**?

בהצלחה!

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים