פרק 4
הפניות

כאשר נוצר עצם, מוקצה לו שטח בזיכרון, שבו נשמרים ערכי התכונות שלו ונתונים נוספים הנדרשים לשימוש בו. כתובת העצם בזיכרון נקראת **הפניה (reference)**. השימוש העיקרי בהפניה הוא כדי להגיע אל העצם ולבקש ממנו לפעול. ההפניה היא ערך שניתן לאחסנו במשתנה וכן להעתיקו ממשתנה למשתנה.
כאשר מוכרז משתנה מטיפוס של מחלקה, אנו נוטים לחשוב שניתן לאחסן בו עצמים שנוצרו מהמחלקה. כך אנו גם מדברים: "c2 הוא דלי מטיפוס המחלקה Bucket". תחושה זו מועילה להבנת תוכניות ונוחה לתקשורת, אך בפועל יש להבין שבמשתנה, מאוחסנת בכל רגע נתון, רק ההפניה לעצם. שימוש במשתנה בתוכנית מתורגם על ידי המערכת, אוטומטית, לשימוש בעצם.

לדוגמה, אחרי ביצוע הפקודה:

```
Bucket b1 = new Bucket (8);
```

נוצר משתנה ששמו b1, נוצר גם עצם "דלי" עם קיבולת 8, **וההפניה** אליו מאוחסנת ב-b1.

נניח כי השורה הבאה היא: `b1.fill (5);`

השימוש בשם המשתנה b1 מתפרש במערכת כשימוש בעצם. סימון הנקודה מתפרש כהודעה לעצם לבצע אחת מפעולותיו. כך פירוש הביטוי הוא הודעה לעצם לבצע את הפעולה עם הפרמטר 5.

בנוסף להפניות לעצמים, יש ערך הפניה מיוחד – ההפניה הריקה, **null**, שהוא גם ערך ברירת המחדל למשתנים המכילים הפניות לעצמים.
אם נכתוב:

```
Bucket b1;
b1.fill (5);
```

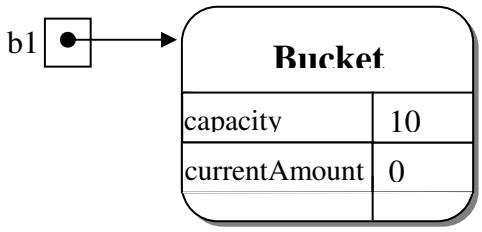
נקבל שגיאת זמן ריצה. הפקודה הראשונה יוצרת משתנה b1, ומאחסנת בו את ההפניה הריקה. הפקודה השנייה היא הוראה לבצע את הפעולה fill של העצם שהפניה אליו מאוחסנת במשתנה. כיון שההפניה הריקה אינה מצביעה על עצם, נקבל שגיאה.
כאשר ההפניה לעצם מאוחסנת במשתנה אנו יכולים להתייחס לשם המשתנה כשם העצם, אבל עלינו לזכור כי שם זה חיצוני לעצם. כמו כן אנו יכולים לחשוב על העצם כמאוחסן במשתנה. כאמור לעיל, זו חשיבה סבירה, אך יש מקרים בהם חשוב להבין כי מה שמאוחסן במשתנה היא ההפניה לעצם ולא העצם עצמו. בהמשך הפרק נסקור מקרים כאלו.

הצבה

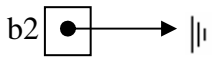
נסתכל על שורות הקוד הבאות:

```
Bucket b1 = new Bucket (10);
Bucket b2;
b2 = b1;
```

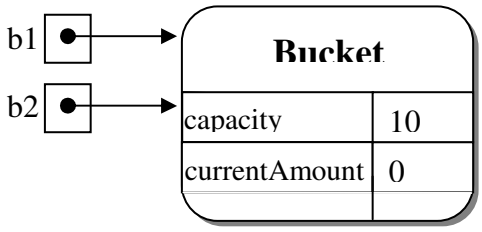
בשורה הראשונה מוצבת במשתנה b1, הפניה למופע של דלי חדש:



בשורה השנייה, מוגדר משתנה בשם b2, בו מאוחסנת הפניה ריקה:



בשורה השלישית, מוצב במשתנה b2 אותו דלי בדיוק שנמצא במשתנה b1:



כיצד יכול אותו עצם להיות מוצב בשני משתנים בו-זמנית? הוא בוודאי אינו יכול להיות מוצב בשני שטחי זיכרון שונים באותו הזמן! עלינו להבין שמשתנה "המכיל" עצם אינו מכיל בשטח הזיכרון שלו את העצם ממש, אלא רק הפניה אל העצם. במשתנים b1 ו-b2 מאוחסנות הפניות אל אותו דלי, וכך יכולים כמה משתנים שונים "להכיל" אותו עצם.

שימו לב שהדלי שב-b1 אינו רק זהה לדלי שב-b2 בכל תכונותיו, אלא שניהם הם **אותו העצם** ממש. שינוי בעצם ש-b1 מפנה אליו, הוא גם שינוי בעצם ש-b2 מפנה אליו, ולהיפך, שכן זהו אותו עצם. כאשר משתמשים בשם המשתנה כשם לעצם, נוכל לומר: אם משנים את כמות המים בדלי b1, משתנה גם כמות המים בדלי b2. למעשה, b1 ו- b2 הם שני שמות לאותו דלי.

? לו היינו רוצים להציב כעת במשתנה b1 דלי אחר, שונה מזה המאוחסן כעת ב-b1 וב-b2, אך עם אותם ערכים, כיצד היינו עושים זאת?

מערכים

בעבר למדתם על מערכים של טיפוסים בסיסיים. למשל על מנת ליצור מערך בגודל 5 של מספרים שלמים כתבתם:

int[] arr = **new** int [5];

בצורה זו נוצר מערך שכל תא בו מאותחל לברירת המחדל של ג׳אווה. במקרה של מספרים שלמים, ברירת המחדל היא 0.

מערך בג׳אווה הוא עצם. יש לו תכונה פומבית בשם length, שערכה אינו ניתן לשינוי (**final**). ערך תכונה זו נקבע בזמן יצירת המערך ואין אפשרות לשנותה לאחר מכן, אורך המערך נשאר קבוע במהלך ביצוע תוכנית. אם אין די מקום במערך נתון, יש ליצור מערך חדש, גדול יותר, ולהעתיק אליו את תוכן המערך.

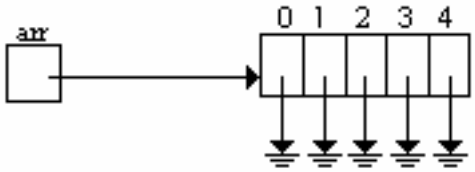
בירור גודל המערך נעשה בעזרת סימון הנקודה כמקובל בעצמים: arr.length

בנוסף לתכונת האורך, שייכים למערך גם התאים שלו, אולם גישה לתאים אלו אינה בעזרת סימון הנקודה אלא בצורת הכתיבה המיוחדת למערכים: arr [i]. כלומר, למרות שבג׳אווה מערכים הם עצמים, הגישה לאיבריהם נכתבת בסגנון המסורתי המקובל גם בשפות שאינן מונחות עצמים.

ניתן ליצור מערך שמכיל עצמים מטיפוס כלשהו. על מנת ליצור מערך של 5 קופסאות נרשום:

Box[] arr = **new** Box [5];

מערך זה מכיל 5 הפניות מטיפוס Box. ברגע יצירת המערך ההפניות הללו מאותחלות לערך ברירת המחדל של ג׳אווה. ערך זה עבור הפניות הוא **null**.



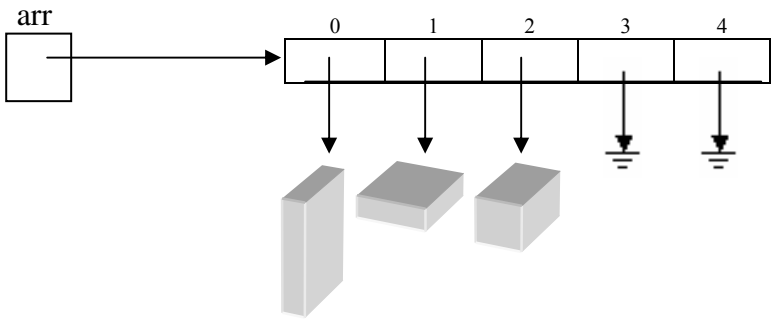
על מנת לאתחל את ההפניות יש צורך ליצור עצמים ולהציב אותם בתוך המשתנים:

arr [0] = **new** Box (1, 2, 5);

arr [1] = **new** Box (3, 2, 1);

arr [2] = **new** Box (2, 2.5, 2);

עתה מוצבים בשלושת התאים הראשונים במערך, עצמים מטיפוס Box.



העברת עצמים לפעולות

ניתן להעביר עצם כארגומנט לפעולה. מה שעובר בפועל אינו העצם אלא הפניה אליו, המאוחסנת בפרמטר המתאים. העברה זו נקראת העברת עצם **על ידי הפניה** (**by reference**).

לדוגמה, נניח שהגדרנו מחלקה בשם Boxes שבכל מופע שלה מוחזקת סדרה של קופסאות במערך arr:

```
public class Boxes
{
        Box[] arr = new Box[10];
}
```

אחת מפעולות המחלקה היא replaceBox (**int** i, Box b), המאפשרת להחליף את הקופסה שבמקום ה-i במערך, בקופסה b. קוד הפעולה נראה כך:

```
public void replaceBox (int i, Box b)
{
        if (0 <= i && i < arr.length)
        {
                this.arr[i] = b;
        }
}
```

נניח כי פעולה זו מופעלת באופן הבא:

```
Boxes myBoxes  = new Boxes();
Box tempBox = new Box (3, 3, 3);
myBoxes.replaceBox (2, tempBox);
```

לאחר ביצוע הפקודה השנייה, המשתנה tempBox מכיל הפניה לקופסה חדשה. בפקודה השלישית מופעלת הפעולה replaceBox: בביצוע הקוד, המשתנה b, הפרמטר הפורמלי של הפעולה, מכיל הפניה לאותה קופסה. (בפרט, זהו מצב בו שני משתנים, tempBox ו-b, מכילים הפניות לאותו עצם.) עם סיום ביצוע הפעולה, המשתנה b, של הפעולה replaceBox, אינו קיים

יותר. עדיין יש שתי הפניות אל הקופסה, אחת במשתנה tempBox, ועוד אחת בתא מספר 2 של המערך arr שהוא תכונה של העצם myBoxes.

פעולה יכולה גם להחזיר עצם. גם כאן, מוחזרת הפניה לעצם ולא העצם עצמו. לדוגמה, נניח שבמחלקה Boxes יש פעולה findLargest, המחזירה את הקופסה בעלת הנפח המקסימלי במערך הקופסאות:

```
public Box findLargest()
{
    קוד למציאת האינדקס של הקופסה בעלת הנפח המקסימלי במערך הקופסאות ...
    return arr[k];
}
```

ההוראה האחרונה בקוד מחזירה **הפניה** לקופסה המאוחסנת במערך בתא מספר k.

כמו באחסון הפניות לעצמים במשתנים כך גם בהעברת הפניות אל פעולות ומהן בחזרה, נוח ומועיל לחשוב במודל פשוט שבו העצמים עצמם מועברים כפרמטרים לפעולות, או מוחזרים מהן. יחד עם זאת חשוב לזכור את המודל האמיתי. בהמשך הפרק נציג מקרים המדגימים מדוע.

הערה: העברת עצמים בשיטת העברת הפניה היא דרך המלך ובג'אווה זו השיטה היחידה.

תכונה שערכה עצם

עד כה דנו במחלקות שהתכונות שלהן היו מטיפוסים פשוטים. אולם, עסקנו גם במחלקות בהן יש תכונה מטיפוס String או מטיפוס מערך. כאמור לעיל, מערכים הם עצמים, וכן הדבר גם לגבי מחרוזות. למעשה, תכונה יכולה להיות מטיפוס של מחלקה כלשהי. מופע של מחלקה שבה מוגדרת תכונה כזו, יקרא **עצם מורכב** (**composite object**). בעצם מורכב, התכונה הזו מכילה הפניה לעצם.

הבה נבנה מחלקה בשם בקבוק-חלב, MilkBottle שמתוארת באופן הבא:

MilkBottle
int capacity **double** fat Date expiryDate
MilkBottle (**int** capacity, **double** fat, Date expiryDate) **int** getCapacity() **double** getFat() Date getExpiryDate()

תכונות המחלקה הן: קיבולת הבקבוק מטיפוס **int**, אחוזי השומן מטיפוס **double** ותאריך התפוגה (תאריך אחרון לשימוש בבקבוק) מטיפוס Date.

מאיזה טיפוס יהיה תאריך התפוגה?

את הטיפוס Date נגדיר בעזרת מחלקה מתאימה. היום, החודש והשנה יהיו תכונותיו של התאריך, ויהיו לו פעולות השימושיות לתאריכים:

Date
int day **int** moth **int** year
Date (**int** day, **int** month, **int** year) **int** getYear() **int** getMonth() **int** getDay() **void** setYear (**int** yearToSet) **void** setMonth (**int** monthToSet) **void** setDay (**int** dayToSet) **boolean** before (Date other) String toString()

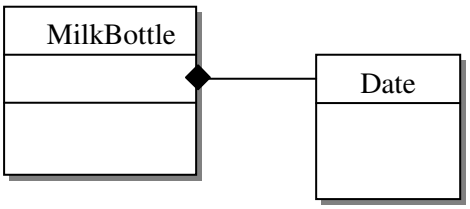
להלן ממשק אפשרי למחלקה Date:

הפעולה בונה עצם מטיפוס Date	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. **הנחה:** ערך הפרמטר הוא מספר שלם לא-שלילי בן ארבע ספרות	**void** setYear (**int** yearToSet)
הפעולה קובעת את ערך החודש על פי הפרמטר הנתון. **הנחה:** ערך הפרמטר הוא מספר שלם בין 1 ל-12	**void** setMonth (**int** monthToSet)

הפעולה קובעת את ערך היום על פי הפרמטר הנתון. הנחה: ערך הפרמטר הוא מספר שלם בין 1 ל-31	**void** setDay (**int** dayToSet)
הפעולה מחזירה **true** אם ורק אם התאריך הנוכחי קודם לתאריך שהועבר אל הפעולה כפרמטר	**boolean** before (Date other)
הפעולה מחזירה מחרוזת המתארת את התאריך בצורה הבאה: <day>/<month>/<year>	String toString()

נזכיר כי **הנחה** בתיעוד הפעולה מודיעה למשתמש במחלקה על הערכים הנכונים שיש לשלוח לפעולה כדי שתפעל כראוי, ומשחררת את ממשמש המחלקה מהצורך לבדוק את העמידה בתנאים אלו בתוך מימוש הפעולה.

את הקשר בין המחלקה MilkBottle למחלקה Date אנו מציינים באופן גרפי בשפת המידול UML, כך:



כלומר, מתקיים כאן קשר של הרכבה, למחלקה MilkBottle **יש** תכונה מטיפוס Date.

נכתוב את המחלקה MilkBottle וכמובן שנקפיד על תיעודה המלא:

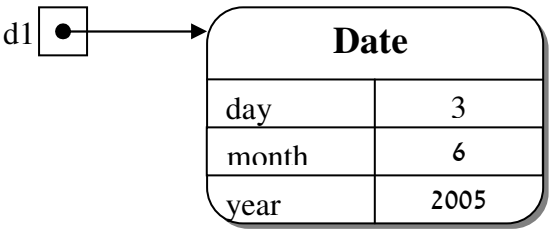
```
/**
 * המחלקה מגדירה בקבוק חלב בקיבולת מסוימת כשהחלב בעל אחוז שומן מסוים
 *  לכל בקבוק חלב יש תאריך תפוגה
 */
public class MilkBottle
{
	private int capacity;

	private double fat;

	private Date expiryDate;

	/**
	 *  פעולה הבונה בקבוק חלב על פי קיבולת, אחוזי שומן ותאריך תפוגה נתונים
	 * @param capacity קיבולת הבקבוק (חייב להיות מספר חיובי)
	 * @param fat אחוזי שומן של החלב (מספר חיובי קטן מ-100.0)
	 * @param expiryDate  תאריך תפוגה
	 */
	public MilkBottle (int capacity, double fat, Date expiryDate)
	{
		this.capacity = capacity;
		this.fat = fat;
		this.expiryDate = expiryDate;
	}

	/**
	 *  הפעולה מחזירה את קיבולת בקבוק החלב
	 */
	public int getCapacity()
	{
		return this.capacity;
	}

	/**
	 *  הפעולה מחזירה את אחוזי השומן של החלב
	 */
	public double getFat()
	{
		return this.fat;
	}

	/**
	 *  הפעולה מחזירה את תאריך התפוגה של הבקבוק
	 */
	public Date getExpiryDate()
	{
		return this.expiryDate;
	}
}
```

שימו לב, כי הגודל, אחוזי השומן והתאריך האחרון לשימוש נשארים קבועים לאורך התוכנית כפי שהיו ברגע יצירת הבקבוק. אין סיבה לכלול פעולות לשינוי התכונות. יתירה מכך, חשוב שלא תהיה דרך לשינוי התכונות כדי למנוע מזייפנים למכור חלב שמן כאילו הוא רזה, או למכור חלב שהתאריך האחרון לשימוש בו עבר. לכאורה, מכיוון שהתכונות מוגדרות כ-**private** ולא נתונות פעולות המשנות את ערכן, באמת אין אפשרות לשנות את תכונות הבקבוק. האם אכן זהו המצב?

נתבונן בפעולה הראשית הבאה:

```
public static void main (String[] args)
{
        Date d1 = new Date (3, 6, 2005);
        MilkBottle mb = new MilkBottle (5, 2.5, d1);
        d1.setYear (2007);
}
```

בשורה הראשונה נוצר עצם מטיפוס Date והוא מוצב בהפניה בשם d1:

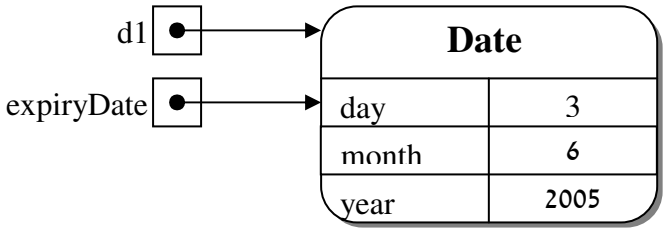


לאחר מכן נוצר עצם מטיפוס MilkBottle והוא מוצב בהפניה בשם mb. העצם הזה קיבל שלושה פרמטרים. שניים מהפרמטרים הם מטיפוס פשוט ואחד הוא מטיפוס Date. עצם זה מועבר על ידי ההפניה שלו המאוחסנת ב-d1.

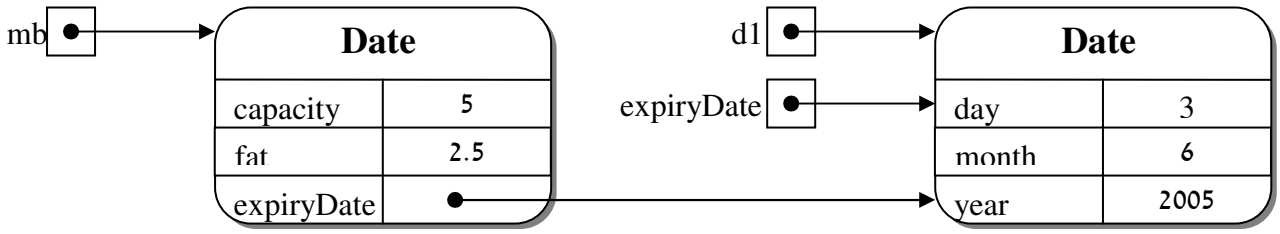
נבחן את הקוד של הפעולה הבונה:

```
public MilkBottle (int capacity, double fat, Date expiryDate)
{
        this.capacity = capacity;
        this.fat = fat;
        this.expiryDate = expiryDate;
}
```

ההפניה בפרמטר d1 (ארגומנט) שהועבר בעת זימון הפעולה, מתוך הפעולה הראשית, מועתקת לפרמטר הפורמלי של הפעולה הבונה expiryDate:

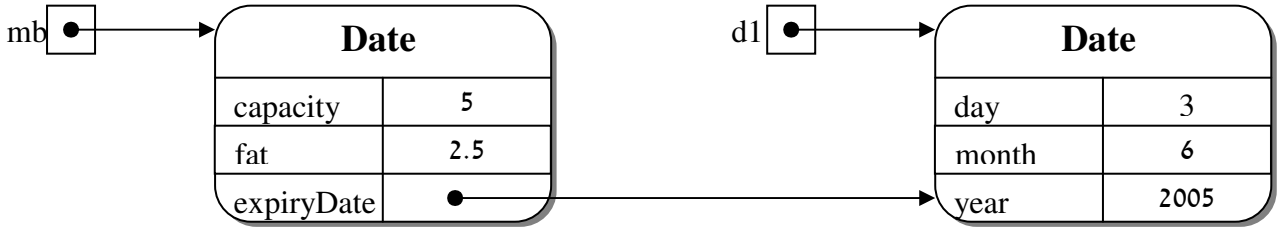


בפקודה האחרונה של קוד הפעולה מועתקת הפניה זו לתכונה expiryDate של העצם:



בנקודה זו קיימות שלוש הפניות לאותו עצם.

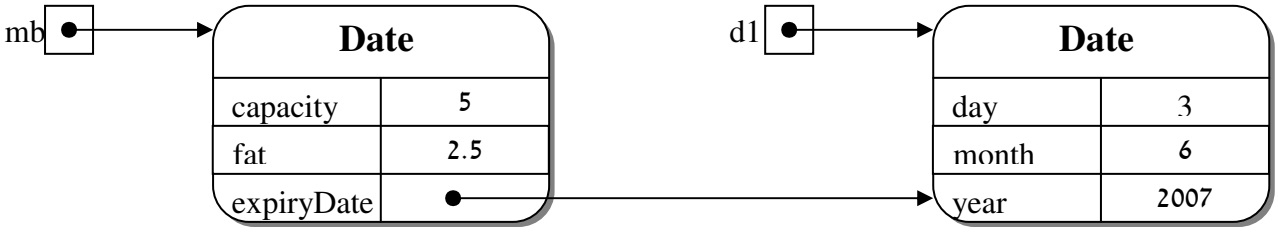
לאחר סיום ביצוע הפעולה-הבונה, המשתנה הזמני expiryDate נעלם, אך המשתנה d1 שהוגדר בפעולה הראשית והתכונה של העצם mb, פונים עדיין לאותו עצם מטיפוס תאריך:



עתה נחזור לשורה השלישית בפעולה הראשית:

```
d1.setYear (2007);
```

שורה זו משנה את תכונת העצם ש-d1 פונה אליו. אך זהו אותו העצם שגם התכונה expiryDate פונה אליו.



כלומר, התאריך האחרון של בקבוק החלב שונה משנת 2005 ל-2007 בלי שמתכנת המחלקה סיפק פעולה לשינוי כזה.
כיוון שהתאריך הוא עצם, הוא נשמר בעזרת הפניה. העתקת הפניה מתוך הפעולה הראשית לתכונה של העצם מאפשרת שינוי התכונה דרך ההפניה שבפעולה הראשית. זהו מצב בלתי רצוי לגבי המקרה המתואר. כיצד נמנע ממנו?
נתבונן שוב בפעולה הראשית המתוארת לעיל. ניתן להשתמש בכתיב מקוצר ולמזג את שתי הפקודות הראשונות בתוכנית. כך יווצר העצם מטיפוס Date תוך כדי הקריאה לפעולה-הבונה:

```
MilkBottle mb = new MilkBottle (2, 3.5, new Date (3, 6, 2005));
```

במקרה כזה, המשתנה היחיד המכיל הפניה לתאריך הוא התכונה expiryDate שבעצם mb. אין דרך לשנותו על ידי פעולות של הבקבוק, ואין הפניה אליו המאוחסנת חיצונית לבקבוק.
אולם, "פתרון" זה מגביל את המתכנת לצורת יצירה מסוימת של עצמים של מחלקת הבקבוקים; יותר מכך אין הוא פתרון שלם שכן הוא אינו מונע לחלוטין את האפשרות לשנות את תאריך התפוגה של בקבוק.

? הסבר כיצד יכול מתכנת המעוניין בכך לשנות את תאריך התפוגה של בקבוק.

? מדוע אין מתכנת יכול לשנות את אחוזי השומן של בקבוק?

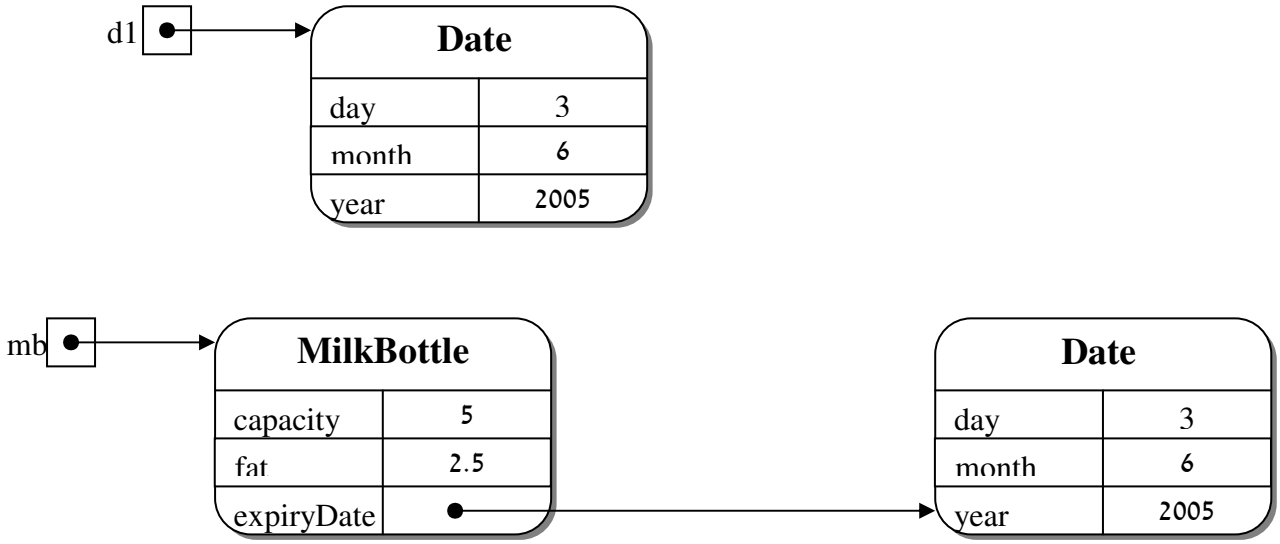
הפתרון המלא לבעיה זו הוא יצירת עותק של התאריך. כך, נציב **עותק** של העצם שאליו פונה d1 בתוך התכונה של הבקבוק ולא את העצם עצמו. להלן אפשרות אחת למימוש הגישה:

```
public MilkBottle (int capacity, double fat, Date expiryDate)
{
        this.capacity = capacity;
        this.fat = fat;
        this.expiryDate = new Date (expiryDate.getDay(),
                                expiryDate.getMonth(), expiryDate.getYear());
}
```

נבחן את ביצוע הקוד של הפעולה הראשית שראינו קודם בהתאם לפעולה-הבונה החדשה:

```
public static void main (String[] args)
{
        Date d1 = new Date (3, 6, 2005);
        MilkBottle mb = new MilkBottle (5, 2.5, d1);
        d1.setYear (2007);
}
```

המצב אחרי שתי השורות הראשונות של הפעולה הראשית יהיה כזה:



הפעם מוצב בתכונה expiryDate של mb, עצם חדש שערכיו זהים לערכי d1. שינוי של d1 בשורה האחרונה של הפעולה הראשית, לא ישפיע על התכונה הזו של העצם.

? בשינוי הפעולה הבונה כמתואר לעיל עדיין לא סתמנו את כל ה"חורים". אילו שינוי או שינויים נוספים בהגדרת המחלקה MilkBottle דרושים כדי להבטיח שאין שום דרך לשנות תאריך תפוגה של בקבוק?

קשיים אחדים מתעוררים בצורת העתקה זו:

1. הבעיה היא כללית ואינה אופיינית לתאריכים בלבד. לתאריך שלוש תכונות בלבד. אם לעצם תכונות רבות, יש צורך בפקודה להעתקת ערכה של כל אחת מהן. במידה ויש לבצע כמה פעולות העתקה במקומות אחדים בתכנית, יש לחזור על סדרת פקודות ההעתקה בכל מקום כזה. דבר זה מוביל להארכה ולסיבוך של התוכנית. גם אין להתעלם מהאפשרות שמתכנת ישגה וישמיט אחת או יותר מפקודות ההעתקה.
2. אם חלק מהתכונות הן פרטיות ולא הוגדרו עבורן פעולות אחזור, לא נוכל לקרוא את ערכיהן ולשלוח אותם לפעולה-הבונה.

אנו נציע פתרון כללי שאין לו את החסרונות הללו והוא הפתרון המועדף. בבסיס הפתרון עומדת ההבחנה שאם סדרת פקודות מהווה פעולה לוגית, כגון במקרה שלפנינו פעולה של "העתקת עצם", ובפרט אם יש שימוש בפעולה זו יותר מפעם אחת, יש לממשה כפעולה. השאלה היחידה שיש לדון בה היא שאלת המיקום של פעולה זו.

פעולה-בונה מעתיקה

כיוון שיצירת עותק של עצם היא פעילות חשובה ונדרשת, טוב לספק בתוך המחלקה שאת עצמיה רוצים להעתיק, פעולת העתקה. כיון שהעתקה כוללת בפרט יצירה של עצם חדש של המחלקה, דבר זה נעשה על ידי הגדרת **פעולה-בונה מעתיקה** (**copy constructor**). פעולה-בונה זו מקבלת כפרמטר עצם מהטיפוס של המחלקה, והיא בונה עצם מאותו טיפוס, שהוא עותק של הפרמטר מבחינת ערכי תכונותיו. על מנת ליצור את העותק, מציבה הפעולה בתכונות העצם הנוצר, את ערכי התכונות של העצם המועבר כפרמטר. זכרו שבתוך המחלקה עצמה ניתן לפנות גם לתכונות פרטיות של כל עצם של המחלקה.

לדוגמה, כיוון שאנו רוצים להעתיק עצמים מטיפוס Date, נספק פעולה-בונה מעתיקה בתוך המחלקה Date:

```
public Date (Date date)
 {
    this.day = date.day;
    this.month = date.month;
    this.year = date.year;
}
```

עם פעולה-בונה זו ניתן ליצור עותק של כל עצם מטיפוס Date:

```
public MilkBottle (int capacity, double fat, Date expiryDate)
{
        this.capacity = capacity;
        this.fat = fat;
        this.expiryDate = new Date (expiryDate);
}
```

השורה:

```
this.expiryDate = new Date (expiryDate);
```

מפעילה את הפעולה-הבונה המעתיקה ויוצרת עותק של הפרמטר. באופן דומה יתבצעו כל ההעתקות הנדרשות בקוד של **MilkBottle**.

? מהם המקומות הנוספים בהגדרת המחלקה MilkBottle שבהם יש לבצע העתקה כדי להבטיח שאין שום דרך לשנות תאריך תפוגה של בקבוק?

האם תמיד יש צורך בהעתקה

עד כה עסקנו בתסריט הקשור לעצם מורכב, שבו כדי להגן על תכונה שהיא עצם שלו אנו מבצעים העתקות, כאשר ערך תכונה זו מועבר אל העצם (למשל בפעולה בונה) או מן העצם (בפעולה מאחזרת). על ידי ביצוע ההעתקות אנו מוודאים שלא יכולה להתקיים מחוץ לעצם המורכב הפניה לתכונה זו.

חשוב לדעת, שהצורך בהעתקה אינו קשור רק לבעיות אלו של שימוש בעצם מורכב. לעתים נרצה ליצור עותק של עצם כדי שיהיו בידינו שני עצמים הזהים בערכי תכונותיהם. לדוגמה, ישנם משחקים שבהם דמויות מסוגלות לשכפל את עצמן. הדמות המשוכפלת, מרגע שנוצרה, היא דמות עצמאית, שאינה תלויה בדמות המקורית. גם אם הדמות המקורית מושמדת, השיבוט שלה יכול להמשיך להתקיים ולפעול. לכן גם בדוגמה זו נשתמש בהעתקה ולא בהפניה נוספת לעצם המקורי.

מצד שני, לא בכל מקרה שנעסוק בעצם מורכב אכן נרצה לבצע העתקה. לעתים נרצה דווקא להשתמש בעצם המקורי. נניח למשל שאנו בונים יישום שבו אנו שומרים פרטים על אנשים ועל קשרי משפחה ביניהם והבסיס לקשרי המשפחה הוא הקשר שבין ילד להוריו. עצם מטיפוס ילד, שומר כתכונה את ההורה שלו. אם נקבל משתנה מטיפוס הורה כפרמטר של הפעולה-הבונה, לא נרצה להעתיק אותו אלא להשתמש בו עצמו. כדי להבין מדוע, נניח שבחרנו לייצג אנשים על ידי שם פרטי, שם משפחה ותאריך לידה. אם יש לנו אוסף גדול של אנשים, יתכן בהחלט שקיימים בו אנשים ששלושת הפרטים האלו זהים עבורם. כאשר אנו מבצעים העתקה של פרטי ההורה ביצירת ילד, כיצד נוכל לקבוע שלשני ילדים אותו הורה? הרי יתכן ששני העצמים שהם ערכי תכונת ההורה שלהם זהים בפרטיהם, אך הם אנשים שונים. אם במקום זאת, נאחסן את העצם המייצג את ההורה ולא עותק שלו, בתכונת ההורה של הילד, נוכל לגלות ילדים של אותו הורה על ידי השוואת העצמים שבתכונות ההורה שלהם: כאשר שני עצמים זהים, הם אותו עצם, הם מייצגים אותו אדם.
תסריט זה מדגים מצב שבו אנו משתמשים בזהות של עצמים (להבדיל משוויון בכל התכונות), כדי לתאר עולם שבו יש חשיבות לזהות. במצב כזה, העתקה אינה רצויה.

? נתונים שני עצמים מאותו טיפוס. כיצד נבדוק שהם זהים, כלומר הם אותו עצם? כיצד נבדוק שכל תכונותיהם בעלות אותו ערך?

כדאי לזכור: הפנייה נוספת לעצם היא למעשה מתן שם נוסף לאותו עצם. לעומת זאת, העתקה היא יצירת עצם חדש, עצמאי, הנמצא במקום אחר בזיכרון.

אין כללי אצבע, המורים מתי כדאי להעתיק עצם ומתי להשתמש במקור. הדבר תלוי בהגדרת הבעיה. בכל מקרה לגופו, צריך לקחת בחשבון את ההשלכות של שימוש בעצם או בעותק.

סיכום

- "עצם מאוחסן במשתנה" פירושו שבמשתנה יש הפניה לעצם.
- הצבת הפנייה שבמשתנה אחד למשתנה אחר יוצרת מצב בו שני משתנים "מכילים" למעשה אותו עצם.
- עצמים המועברים כפרמטרים לפעולות עוברים בשיטת העברת הפניה - by reference. ערך ההפניה מועתק לפרמטר (שהוא משתנה מקומי), כך שנוצרות שתי הפניות לאותו עצם. פעולה שתעשה דרך ההפניה המקומית תשנה את העצם. את השינוי ניתן יהיה לזהות על ידי גישה אל העצם מהמשתנה החיצוני.
- עצם מורכב הוא עצם שלפחות אחת מתכונותיו היא עצם.
- בשימוש בעצם מורכב תתכן פגיעה בהכמסת התכונה שהיא עצם.
- פעולה-בונה המייצרת עותק של העצם שהיא מקבלת כפרמטר נקראת פעולה-בונה מעתיקה (copy constructor).
- ההבחנה בין הפעולות הבונות השונות נעשית כזכור על פי השוני ברשימת הפרמטרים שבכותרתן.
- בכל בעיה כדאי לשקול האם רוצים להשתמש בעותק של עצם או בעצם עצמו בהתאם לתנאי הבעיה.

מושגים

העברת הפניה	by reference
עצם מורכב	composite object
פעולה בונה מעתיקה	copy constructor

פרק 4 דף עבודה מס׳ 1

פעולה-בונה מעתיקה

מטרות

תרגול פעולה-בונה מעתיקה (copy constructor).

מה עליכם לעשות:

1. הוסיפו למחלקה Point שהגדרתם בדף עבודה מס׳ 1 של פרק 3, פעולה בונה מעתיקה (copy constructor).
2. לפניכם הפעולה הראשית הבאה:

```
public static void main (String[] args)
{
        int x = IO.readInt();
        int y = IO.readInt();
        Point p1 = new Point (x, y);
        Point p2 = p1;
        Point p3 = new Point (p1);
        p2.setX (5);
        p3.setY (8);
        System.out.println (p1);
        System.out.println (p2);
        System.out.println (p3);
}
```

שאלות

1. עקבו בעזרת ציור אחר ביצוע התוכנית עבור הקלט x=4,y=9. הריצו את התוכנית, ובדקו את תשובותיכם.
2. כמה עצמים מטיפוס Point נוצרו בתוכנית הראשית הזו? הסבירו.

בהצלחה!

פרק 4 דף עבודה מס׳ 2

ניקח נקודות ונבנה מלבן

מטרות

תרגול של עצמים מורכבים.

חשיבותה של הפעולה-הבונה מעתיקה.

חשיפה לרעיון של היכולת לשנות ייצוג של מחלקה ללא שינוי הממשק.

המחלקה Rectangle

המחלקה Rectangle מגדירה מלבן שצלעותיו מקבילות לצירים. להלן ממשק המחלקה:

הפעולה בונה מלבן על פי הפרמטרים הנתונים: הנקודה השמאלית התחתונה, הנקודה הימנית העליונה	Rectangle (Point bottomLeft, Point topRight)
הפעולה בונה מלבן על פי הפרמטרים הנתונים: הנקודה השמאלית התחתונה, רוחב המלבן, גובה המלבן	Rectangle (Point bottomLeft, **int** width, **int** height)
הפעולה מחזירה את השטח של המלבן	**int** getArea()
הפעולה מחזירה את ההיקף של המלבן	**int** getPerimeter()
הפעולה מזיזה את המלבן: • על ציר X לפי deltaX: ערך חיובי – הזזה ימינה, ערך שלילי – הזזה שמאלה. • על ציר Y לפי deltaY: ערך חיובי – הזזה למעלה, ערך שלילי – הזזה למטה	**void** move (**int** deltaX, **int** deltaY)
הפעולה מחזירה מחרוזת המתארת את המלבן בצורה Rectangle: bottom-left point = (<X> , <Y>) top-right point = (<X> , <Y>)	String toString()

שימוש בפעולות המלבן לדוגמה:

ניצור מלבן על פי הנתונים של שני הקודקודים המגדירים אותו:

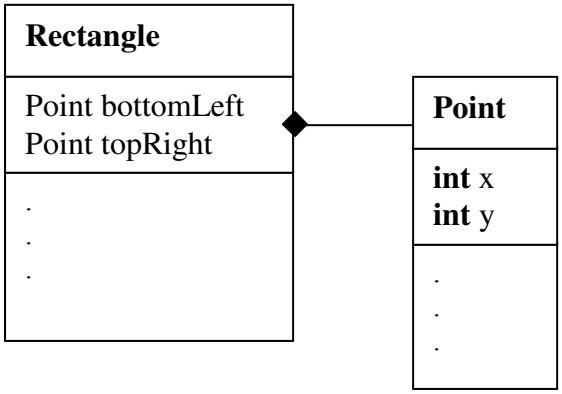
bottomLeft = (2,1) topRight = (7,5)

לאחר הפעלת הפעולה: move(-1, 2), כל הקודקודים ישתנו באותו יחס, והמלבן יזוז על מערכת הצירים: שמאלה ביחידה אחת ולמעלה בשתי יחידות.

נתוניו החדשים של המלבן יהיו:

bottomLeft = (1,3) topRight = (6,7)

שימו לב: המלבן הינו עצם מורכב שתכונותיו הן מטיפוס נקודה:



כדי לבצע את התרגיל יהיה עליכם להשתמש במחלקה Point המורחבת (הכוללת פעולה-בונה מעתיקה) אשר הגדרתם בדף העבודה הקודם. הוסיפו אותה לפרויקט.

מה עליכם לעשות?

1. בחרו ייצוג למחלקה מלבן.
2. כתבו את המחלקה Rectangle, בעלת הממשק המתואר לעיל. אל תשכחו לתעד את המחלקה באופן מלא.
3. כתבו תוכנית בדיקה קצרה שתוכיח את תקינות המחלקה Rectangle, על פי ההנחיות הבאות:

המחלקה TestRectangle

מחלקה זו תשמש כתוכנית בדיקה ותאפשר לכם לבדוק את המחלקה Rectangle שכתבתם. המחלקה תכלול את הפעולה הראשית main(...), ובה יהיה עליכם לבצע את הדברים הבאים:

1. ליצור מלבן על פי נתונים של שני קודקודים.
2. ליצור מלבן נוסף על פי הנתונים (x,y) של הקודקוד השמאלי התחתון, רוחב המלבן וגובהו.
3. להדפיס את תיאור המלבנים.
4. להזיז את אחד המלבנים למעלה ושמאלה, ואת המלבן האחר ימינה ולמטה.
5. להדפיס את תיאור המלבנים לאחר התזוזות.

שאלה:

א. אילו תכונות בחרתם למחלקה Rectangle? האם ישנה אפשרות נוספת?

ב. האם יש צורך להשתמש בפעולה בונה מעתיקה ביצירת המלבן? אם כן, הסבירו היכן ומדוע?

הערה:

1. יש ליצור מלבנים עם נתונים תקינים (אין צורך לבצע בדיקות תקינות).

בהצלחה!

פרק 4 דף עבודה מס' 3

נוסע ודרכון

מטרות

- תרגול יסודי של עצמים מורכבים.
- תרגול נוסף של פעולה-בונה מעתיקה (copy constructor).

מה עליכם לעשות?

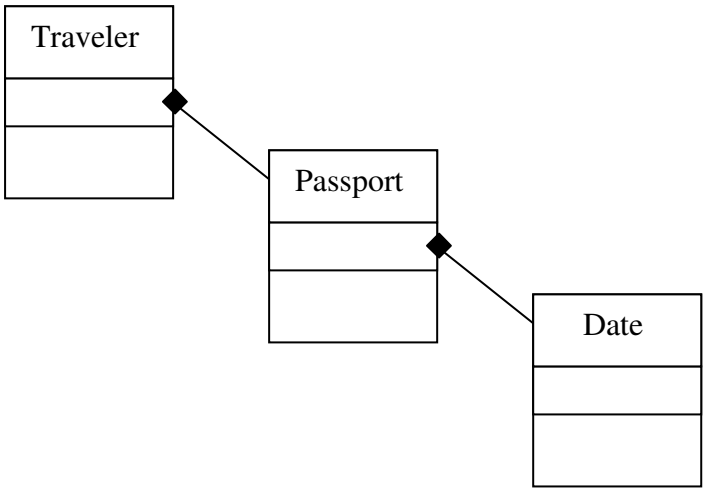
בדף עבודה זה עליכם לממש את המחלקות הבאות:

Passport – מגדירה דרכון של נוסע.

Traveler – מגדירה נוסע.

כדי לממש את המחלקות השתמשו במחלקה Date, שכתבתם בפרק 3. עליכם להוסיף לה פעולה-בונה מעתיקה (copy constructor).

היחסים והקשרים בין המחלקות מתוארים באיור הבא:



כזכור, הסימון a ◆⟵ b מייצג קשר של הרכבה (b מורכב מ- a).

המחלקה Passport

מחלקה זו מגדירה דרכון של נוסע. לשם מימושה, עליכם להשתמש במחלקה Date.

הפעולה בונה דרכון חדש על פי הנתונים המועברים אליה: שם, מספר דרכון ותאריך תפוגה	Passport (String name, **int** number, Date expiryDate)
פעולה-בונה מעתיקה	Passport (Passport passport)
הפעולה מחזירה מחרוזת המתארת את הדרכון (על פי המבנה המתואר למטה)	String toString( )
הפעולה מחזירה **true**, אם ורק אם הדרכון תקף בתאריך הנתון	**boolean** isValid (Date dateChecked)
הפעולה מעדכנת את תאריך התפוגה של הדרכון	**void** setExpiryDate (Date ExpiryDate)

המחרוזת המתארת את הדרכון צריכה להיראות כך:

Name: <name>

Pass. num: <number>

Exp date: <expiry date>

המחלקה Traveler

מחלקה זו מגדירה נוסע. פרטיו של כל נוסע הם הדרכון שלו והמידע האם הנוסע שילם או לא. שימו לב, על פי היחסים שהוגדרו באיור 1, למחלקה Traveler תכונה שהיא מופע של המחלקה Passport.

הפעולה בונה נוסע חדש על פי הנתונים	Traveler (Passport passport, **boolean** hasPaid)
הפעולה מאפשרת לנוסע לשלם עבור הנסיעה	**void** pay()
הפעולה מחזירה **true**, אם ורק אם הנוסע שילם עבור הנסיעה	**boolean** hasPaid()
הפעולה מחזירה מחרוזת המתארת את הנוסע. המחרוזת תהיה זהה לגמרי למחרוזת המתארת את דרכון הנוסע	String toString( )

boolean checkTravel (Date travelDate)	הפעולה מחזירה **true**, אם ורק אם הנסיעה אפשרית בתאריך הנתון. הנסיעה אפשרית אם דרכון הנוסע תקף בתאריך הנתון, ואם הנוסע שילם

בדיקת התוכנית

כדי לבדוק את נכונות התוכנית כתבו מחלקת בדיקה קצרה שתיצור מספר עצמים מטיפוס נוסע, ותדפיס אותם רק אם הנסיעה שלהם אפשרית בתאריך 1 בינואר שנת 2007.

בהצלחה!

פרק 4 דף עבודה מס׳ 4

תרגיל תיאורטי

מטרות

תרגול תיאורטי של הנושאים שנלמדו בפרק.

מה עליכם לעשות?

1. השלימו את כותרות הפעולות של המחלקה A:

הפעולה הבונה מקבלת את הפרמטר: **double** x	
פעולה-בונה ללא פרמטרים, מאתחלת את x לאפס	
פעולה-בונה מעתיקה	
הפעולה מחזירה את הערך של x	
הפעולה משנה את הערך של x לפי הפרמטר	
הפעולה מחזירה מחרוזת המתארת את A בצורה הבאה: A : <x>	

2. כתבו את המחלקה A.
3. מהן התכונות החייבות להופיע במחלקה A? נמקו את תשובותיכם.
4. מהו מספר הפעולות-הבונות של המחלקה? כיצד נקרא המנגנון שמאפשר יותר מפעולה-בונה אחת?
5. לפניכם תוכנית ראשית המשתמשת במחלקה A:

```
public static void main (String[] args)
{
        A a1 = new A (7);
        A a2 = new A (a1);
        A a3 = a2;
        a1.setX (10);
        a3.setX (5);
        System.out.println (a1);
        System.out.println (a2);
        System.out.println (a3);
}
```

בתוכנית זו השתמשנו בשם setX עבור הפעולה המשנה את ערך x. מותר לכם להשתמש בשם אחר כפי שבחרתם בסעיף 1.

- מה יודפס בתום הרצת התוכנית.
- כמה עצמים מטיפוס A נוצרו בפעולה הראשית? הסבירו.

6. לפניכם קוד המחלקה B:

```
public class B
{
	private A a;

	public B (A a)
	{
		this.a = a;
	}
	public A getA()
	{
		return this.a;
	}
	public String toString()
	{
		return "B:" + this.a.toString();
	}
}
```

בתוכנית הראשית קיימת הפעולה הבאה:

```
public static void main (String[] args)
{
	A a1 = new A (10);
	B b = new  B(a1);
	a1.setX (7);
	A a2 = b.getA();
	a2.setX (13);
	System.out.println (a1);
	System.out.println (a2);
	System.out.println (b);
}
```

מה תהיה תוצאת ההדפסה?

7. תקנו את הקוד של המחלקה B כך שההדפסה של התוכנית הראשית שלמעלה תהיה:

```
A: 7
A: 13
B: A: 10
```

8. הסבירו בקצרה מהן הבעיות הקיימות כאשר הפניות בשני משתנים פונות לאותו מקום בזיכרון.

בהצלחה!