

פרק 6

הפניות ועצמים מורכבים

כאשר נוצר עצם, מוקצה לו שטח בזיכרון, ובו 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.

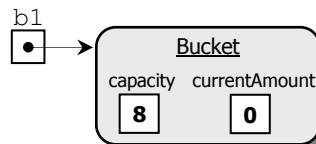
כאשר מוכרז משתנה מטיפוס מחלקה, אנו נוטים לחשוב שניתן לאחסן בו עצמים שנוצרו מהמחלקה. כך אנו גם מדברים: "c2 הוא דלי מטיפוס המחלקה Bucket". תחושה זו מועילה להבנת תוכניות ונוחה לתקשורת, אך בפועל יש להבין שבכל רגע נתון מאוחסנת במשתנה רק ההפניה לעצם, כלומר כתובתו בזיכרון, ולא הוא עצמו. שימוש במשתנה בתוכנית מתורגם על ידי המערכת באופן אוטומטי לשימוש בעצם.

בהמשך הפרק נציג דוגמאות ומצבים שיסבירו וידגימו מדוע הטענה שבמשתנה נמצאת רק הפניה לעצם ולא העצם עצמו היא טענה נכונה, ומדוע חשוב להבינה.

א. אתחול של משתנה מטיפוס מחלקה

האתחול הפשוט ביותר של משתנה מטיפוס מחלקה נעשה כך:

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



באגף שמאל מוגדר משתנה ששמו b1 מטיפוס המחלקה Bucket. באגף ימין נוצר עצם "דלי" עם קיבולת 8. **ההפניה** לדלי החדש מאוחסנת ב-b1.

נניח שהשורה הבאה בקוד תהיה:

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

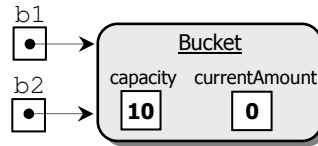
השימוש בשם המשתנה b1 מחזיר את ההפניה המאוחסנת במשתנה. הנקודה אחרי שמו של המשתנה מתפרשת במערכת כהוראה לגשת אל העצם שההפניה מפנה אליו, ולהשתמש בו. לפיכך, פירוש הביטוי b1.fill(5) הוא הודעה לעצם לבצע את הפעולה fill(...) על הפרמטר 5. אפשרות נוספת לאתחול משתנה מטיפוס מחלקה היא על ידי הצבה של הפניות.

נסתכל על שורות הקוד האלה:

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

בשורה הראשונה מוצבת במשתנה b1 הפניה למופע חדש של דלי, שהקיבולת שלו היא 10. בשורה השנייה, מוגדר משתנה בשם b2, שאליו מועתקת ההפניה הקיימת ב-b1. כלומר כעת קיימות שתי הפניות לאותו העצם. במשתנים b1 ו-b2 מאוחסנות הפניות לאותו הדלי.

תרשים העצמים שלפניכם מתאר את מצב המשתנים בסוף קטע הקוד :



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, ולהפך, שכן זהו אותו עצם.

נראה למשל את הקוד הזה :

```
b1.fill(3);
b2.fill(2);
System.out.println(b1);
System.out.println(b2);
```

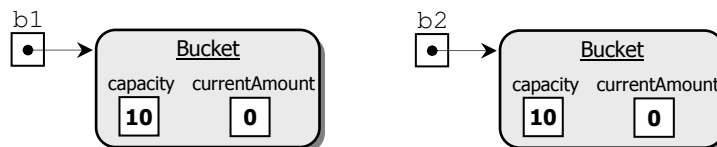
מהן כמויות המים בדליים?

מכיוון ששתי ההפניות פונות לאותו הדלי, העדכונים בפעולות הראשונות התבצעו על דלי יחיד זה, ולכן בסוף התהליך יהיו בדלי חמישה ליטרים מים. אם תריצו את הקוד יוכיחו ההדפסות שאכן כמויות המים בדליים שוות.

אם נרצה להציב בשני המשתנים דליים השווים בערכיהם אך שונים בזהותם, יהיה עלינו לבצע את הקוד שלפניכם, המשתמש בפעולה בונה מעתיקה :

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

במקרה זה נוצר עצם נוסף שהוא העתק מדויק של העצם שאליו פונה b1. ההפניה אל העצם החדש מוצבת ב-b2 :



דרך נוספת לאתחול משתנים מטיפוס מחלקה הוא על ידי הצבת הערך המיוחד: null. ערך זה הוא הפניה ריקה שאינה מובילה לשום עצם. ניתן להציב ערך זה בכל משתנה מטיפוס מחלקה כך :

```
Bucket b1 = null;
```

ניסיון לפנות אל עצם דרך משתנה שבו מוצבת ההפניה null יגרור שגיאה בזמן הריצה.

לסיכום: משתנים מטיפוס מחלקה מכילים הפניות לעצמים או `null`. השמה שמתבצעת בין משתנים כאלה, מעתיקה את ההפניה ממשתנה אחד לאחר. אם מדובר במשתנים המפנים לעצם (ולא `null`), כמה משתנים יפנו לאותו העצם בזיכרון. אם משנים את העצם דרך אחת ההפניות, אזי ניתן לראות את השינוי דרך יתר ההפניות.

ההבחנה שמשתנה מכיל הפניה ולא עצם אינה מובנת מאליה. במהלך הפרק תגלו שדברים שנראים לא מובנים בהתחלה מתבארים כשזוכרים בהבחנה זו. חשיבותו של נושא זה רבה, ועל כן הוא יידון ויודגם עוד רבות במהלך הפרק.

ב. משתנה מטיפוס מחלקה שאינו מאותחל

התבוננו בקטע הקוד שלפניכם:

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

הפקודה הראשונה מגדירה משתנה `b1`, שאינו מאותחל. הפקודה השנייה היא הוראה לבצע את הפעולה `fill(...)` של העצם שהפניה אליו מאוחסנת במשתנה. מכיוון שהעצם לא אותחל, הקומפילציה תיעצר, ואנו נקבל שגיאת הידור.

שימו לב שמצב זה של "אי-אתחול משתנה מטיפוס מחלקה" שונה ממצב שבו משתנה כזה מכיל `null`, ואכן סוגי השגיאות שיקרו בעת הניסיון לבצע: `b1.fill(...)` בשני המצבים הם שונים.

ג. השוואת הפניות

נניח כי קיימים שני דליים `b1` ו-`b2`. נסתכל על תנאי השוויון הזה:

```
if (b1 == b2)...
```

מה נובע מההסברים שנתנו על משתנים מטיפוס מחלקה לגבי משמעותו של תנאי זה?

ייתכן שנחשוב שהתנאי בודק האם ערכי התכונות של שני הדליים שווים, אך זוהי גישה שגויה. התנאי משווה את ההפניות בשני המשתנים. כאשר אנו משווים הפניות אנו בודקים למעשה אם מדובר באותו עצם, וזאת אנו עושים באמצעות העלאת השאלה האם בפועל שני המשתנים מכילים הפניות לאותו העצם.

נתבונן בקטע שלפנינו:

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

```
if (b1 == b2)
```

```
    System.out.println("שני המשתנים מפנים לאותו עצם");
```

```
else
```

```
    System.out.println("שני המשתנים מפנים לשני עצמים שונים");
```

כיוון שמשתנים המכילים הפניות שווים רק אם הם פונים לאותו העצם, תנאי השוויון בין ההפניות אינו מתקיים, וערכו הוא **false**. ההדפסה שתתקבל היא: "שני המשתנים מפנים לשני עצמים שונים".

? כתבו קטע קוד המחזיר **true** אם **b1** ו-**b2** שווים בערכי תכונותיהם, ואחרת הוא מחזיר **false**.

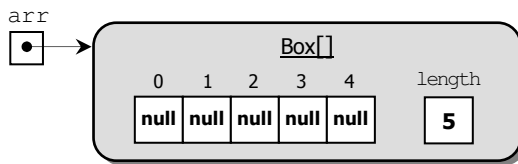
ד. מערך של עצמים

בפרק 2 הכרנו מערך המכיל עצמים. מאז הגדרנו כללי איור ורכשנו ידיעות משלימות. כעת אנו יכולים להציג מערך של עצמים בעזרת תרשים מפורט. מערך הוא עצם, לכן כל תאי המערך מאותחלים ברגע בנייתו לערך ברירת המחדל של טיפוס האיבר שלו. ערך ברירת המחדל לעצמים הוא **null**.

כדי ליצור מערך של חמש קופסאות, נכתוב:

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

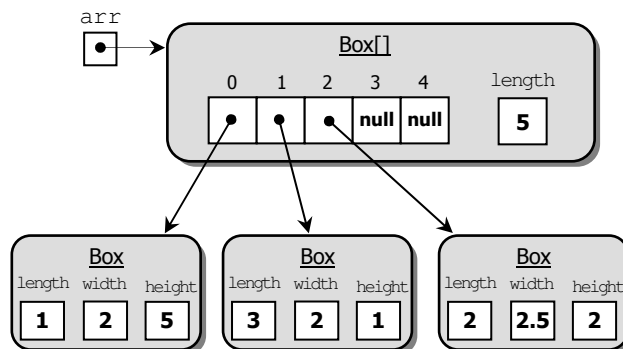
המשתנה **arr** מכיל הפניה למערך, ובו 5 הפניות מטיפוס **Box**. ברגע יצירת המערך, ההפניות הללו מאותחלות לערך ברירת המחדל של ג'אוה (**null**):



כדי לאתחל את המערך יש צורך ליצור עצמים ולהציב את ההפניות אליהם בתוך תאי המערך:

```
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**:

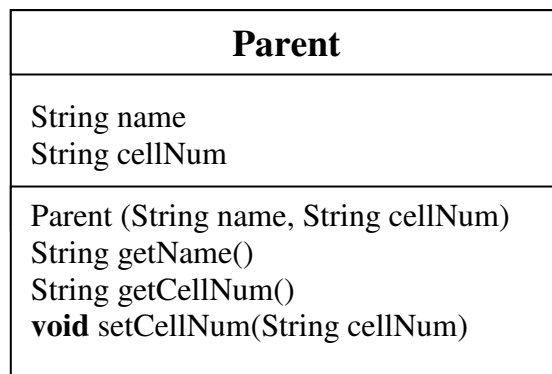


ה. עצם מורכב

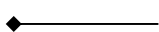
תכונה של מחלקה יכולה להיות מטיפוס כלשהו – טיפוס פשוט או טיפוס של מחלקה. כאשר תכונה אחת או יותר של מחלקה היא מטיפוס מחלקה כלשהי, אנו קוראים לעצם הנוצר מהמחלקה **עצם מורכב (composite object)**.

כדי להסביר את השימוש בעצמים מורכבים נשתמש בדוגמה אחת לאורך היחידה, והיא: יישום של מערכת בית ספרית. נקדים ונאמר כי אף שאנו מנסים להציג יישום אמיתי, הרי אין באפשרותנו לעסוק בכל הדרישות והאילוצים האמיתיים שמציבה מערכת שכזו, ולכן אנו נציב הסתייגויות ודרישות מיוחדות ולעתים בלתי מציאותיות, כך שנוכל להתמקד במצבים המעניינים אותנו.

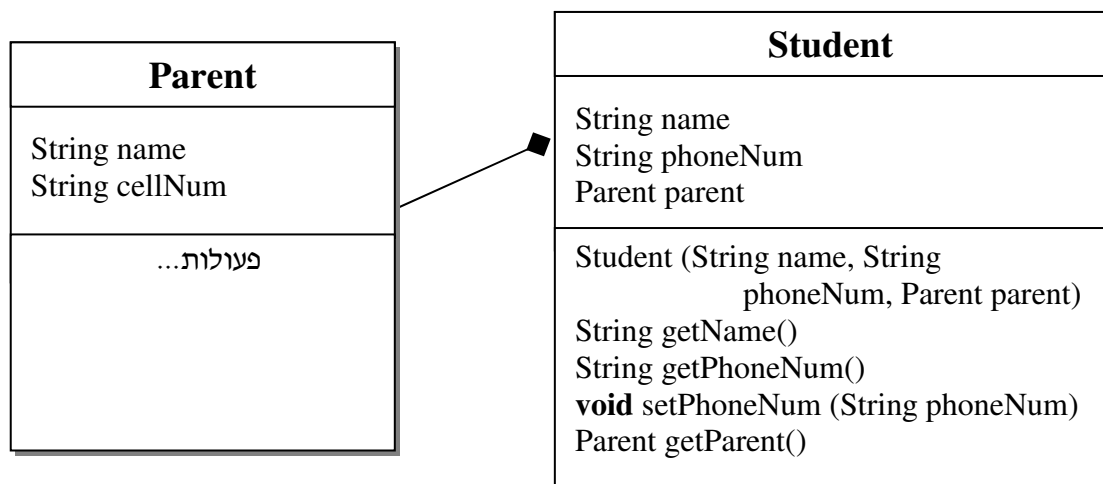
בסעיף זה נתמקד ברישום תלמידים. השלב הראשון ברישום התלמידים הוא הקמת מאגר הורים. עבור כל הורה, Parent, נשמרים שמו המלא ומספר הטלפון הסלולרי שלו.



לאחר מכן ניתן לרשום את התלמידים. עבור כל תלמיד, Student, ייבנה עצם המכיל את פרטיו: שמו המלא, מספר הטלפון בביתו ושמו של ההורה האחראי עליו מבחינת בית הספר. שם ההורה האחראי יילקח ממאגר ההורים.

בין שתי המחלקות קיים יחס של הרכבה. יחס זה מוצג בסימוני UML כך: : ופירושו של סימן זה הוא שלמחלקה שאליה פונה המעוין יש תכונה מטיפוס המחלקה שבקצהו האחר של הקו.

בדוגמה שלנו, נאמר שלמחלקה Student יש תכונה מטיפוס Parent:



ה.1. ייצוג ומימוש המחלקה המגדירה עצם מורכב

נתבונן בהגדרת התכונות של המחלקה Student :

```
public class Student
{
    private String name;
    private String phoneNum;
    private Parent parent;
}
```

בדוגמה המלווה אותנו, סביר שאתחול הערכים של העצם המורכב ייעשה בעזרת זימון פעולה בונה, למשל זו שהקוד שלה לפניכם :

```
public Student(String name, String phoneNum, Parent parent)
{
    this.name = name;
    this.phoneNum = phoneNum;
    this.parent = parent;
}
```

ערכיו של העצם המורכב נקבעים על פי הפרמטרים המועברים לפעולה הבונה. הפרמטר האחרון הוא מטיפוס Parent. כלומר שכדי ליצור עצם מטיפוס Student, עלינו לשלוח כפרמטר (אחרון) אל הפעולה הבונה של Student עצם מטיפוס Parent שהוגדר קודם לכן. העברת העצם מטיפוס הורה נעשית על ידי העברת ההפניה אליו, כפרמטר לפעולה.

נתאר את תהליך הרישום המתנהל במשרדה של המזכירה. הורה ובנו התלמיד הגיעו למשרד וביקשו להירשם לבית הספר. פרטיו של ההורה נרשמו במאגר ההורים :

```
Parent p1 = new Parent("Katz Meir", "054-2345678");
...
```

בשלב שני ניתן לרשום את התלמיד החדש. זימון הפעולה הבונה ייראה כך :

```
Student st1 = new Student("Katz Oren", "02-8765432", p1);
```

לאחר ביצוע המהלך המתואר נוספו למאגרי המידע המתאימים במזכירות בית הספר עצמים המתארים את ההורה ואת התלמיד.

למחלקה Student יהיו גם פעולות נוספות הקובעות חלק מערכי התכונות ומאחזרות אותן, ובפרט את ההורה :

```
public Parent getParent()
{
    return this.parent;
}
```

אנו נמנע מהכללת הפעולה setName(...) במחלקה Student, כשם שנמנענו מלהוסיפה למחלקה Parent. בהמשך הפרק נסביר את הסיבה להשמטת פעולה זו.

ניתן לגשת לתכונות הורה של תלמיד דרך ההפניה אל ההורה הקיימת בעצם מטיפוס Student:

```
Parent p = st1.getParent();
String parentName = p.getName();
```

ובשורה אחת, בקיצור: `String parentName = st1.getParent().getName();`

כדי להבין את משמעות ההפניות, נבחן במפורט את המתרחש כאשר משתמשים בקוד שהצגנו עד כאן.

ה.2. הפניה המועברת כפרמטר

נתייחס לקוד הפעולה הבונה:

```
public Student(String name, String phoneNum, Parent parent)
{
    this.name = name;
    this.phoneNum = phoneNum;
    this.parent = parent;
}
```

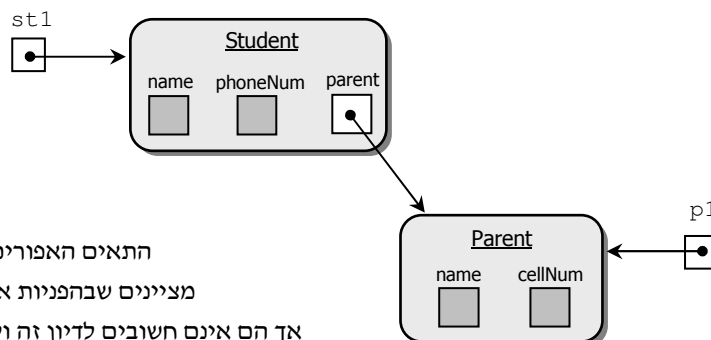
אנו רואים כי הפעולה הבונה של Student קיבלה פרמטר שהוא הפניה לעצם, מטיפוס Parent. המשתנה parent שהוגדר בכותרת הוא משתנה מקומי של הפעולה.

בעת הזימון:

```
Parent p1 = new Parent("Katz Meir", "054-2345678");
...
Student st1 = new Student("Katz Oren", "02-8765432", p1);
```

מקבל משתנה זה את הערך שהועבר אליו, שהוא הפניה לעצם מטיפוס Parent.

בגוף הפעולה הבונה מתבצעת השמה של ההפניה שבמשתנה parent לתוך התכונה parent של התלמיד. לאחר שהפעולה הבונה הסתיימה, הפרמטר parent אינו קיים יותר, אך התכונה parent של התלמיד מכילה הפניה לעצם של ההורה. בשלב זה במהלך ריצת התוכנית המשתנה p1 והתכונה parent פונים לאותו העצם:



התאים האפורים בתרשים העצמים מציינים שבהפניות אלה מוצבים ערכים, אך הם אינם חשובים לדיון זה ולכן אינם מפורטים)

? כתבו קטע קוד שיוכיח או יסתור את הטענה ששתי ההפניות: p1 והתכונה parent של st1, מכילות הפניות לאותו העצם ואינן מכילות עצמים נפרדים (הזהים בערכי תכונותיהם).

הערה חשובה: נזכיר את ההנחה הסמויה המלווה אותנו לכל אורך יחידת הלימוד: הפניה לעצם, המועברת כפרמטר, אינה שווה null והיא מפנה תמיד לעצם קיים. לכן איננו צריכים לבדוק בראשית המימוש האם ההפניה שווה null. רק במקרים חריגים שבהם נרצה שערך ההפניה יהיה שווה null, נוסיף תיעוד מתאים ובדיקה בגוף הפעולות.

ה.3. הפניה כערך החזרה

הפעולה getParent() מוגדרת במחלקה Student ומחזירה הפניה לעצם מטיפוס Parent:

```
public Parent getParent()
{
    return this.parent;
}
```

נבחן בעזרת הקוד שלפניכם מה משמעות הפניה זו:

```
Parent p1 = new Parent("Katz Meir", "054-2345678");
...
Student st1 = new Student("Katz Oren", "02-8765432", p1);
Parent p2 = st1.getParent();
...
p2.setCellNum("052-6782345");
...
```

? ציירו תרשים עצמים המתאר את העצמים הקיימים בקטע הקוד המופיע למעלה ואת מצבם בסוף הקטע.

רעיון זה של הפניות שונות לאותם העצמים יכול לסייע בידינו במצבי אמת. במקרים רבים נרשמים אחים בני משפחה אחת לאותו בית הספר. במקרה שכזה נהיה מעוניינים שהעצמים שהוגדרו עבור האחים יכילו הפניה לאותו ההורה האחראי.

למשל במקרה שנרצה לעדכן את מספר הטלפון הסלולרי של ההורה דרך אחד האחים, ומכיוון שמדובר בהורה משותף של כמה אחים, נוח לעדכן את פרטיו על ידי גישה לעצם המייצג אחד מהם ולדעת שהעדכון משתקף גם אצל יתר האחים. אם ניגש אל ההורה דרך אח אחר העדכון ייראה גם משם. בהמשך הפרק נראה דוגמאות אחרות בהן עדכון כזה, על ידי "יד נעלמה", אינו רצוי ואנו נהיה מעוניינים בניתוק הקשר בין העצמים.

לסיכום: משתנים שונים יכולים להחזיק הפניות לאותו העצם. במקרה כזה שינוי שיעשה בעצם על ידי שימוש בהפניה שבאחד המשתנים ייראה אחר כך בכל אחת מההפניות האחרות.

פרמטרים וערכי החזרה מטיפוס מחלקה הם למעשה הפניות. כך – העברת עצם כפרמטר לפעולה, פירושה למעשה הצבת עותק של ההפניה לעצם בפרמטר של הפעולה, למשך ביצוע הפעולה (הפרמטר מתבטל בסוף הפעולה). החזרת עצם מפעולה, פירושה החזרת ההפניה אליו.

1. עצם מורכב המייצג אוסף

נרחיב את דוגמת היישום של המערכת הבית ספרית, כך שתאפשר התייחסות לקבוצות של תלמידים לפי: שכבה, מחזור, כיתה. ניצור מחלקה בשם StudentList שתשמש כתבנית לייצור רשימות כיתתיות של תלמידים. לפניכם ממשק המחלקה המאפשר הכנסת תלמיד לרשימה כיתתית, הוצאת תלמיד מהרשימה, קבלת פרטיו של תלמיד על פי שמו והדפסת דף קשר כיתתי ממוין בסדר אלפביתי. היישום כולו מורכב (נכון לעכשיו) מהיחידות הבאות: המחלקות Parent ו-Student, מאגר הורים ומאגר תלמידים שבפרטיהם לא דנו, ומחלקה חדשה StudentList.

כפי שציינו בתחילת הפרק, ביישום המערכת הבית ספרית אנו מתייחסים לעולם מוקטן ומוגבל. בעולם זה אדם מזוהה על פי שמו ולא ייתכן שיהיו קיימים שמות כפולים. כך נחסוך את העיסוק בהנחות בתיעוד ובבדיקות בתוך הקוד. כמו כן אנו מניחים שלמזכירה יש דרך פשוטה לדעת אם עוד יש מקום להכניס תלמיד נוסף לכיתה, בלי שנוסיף למחלקה תכונה ופעולות חדשות.

ממשק המחלקה StudentList

המחלקה StudentList מגדירה קבוצה של תלמידים הנקראת "רשימה כיתתית". תלמיד ברשימה מזוהה על פי שמו (אין בכיתה תלמידים בעלי שם זהה). אין צורך לבדוק האם קיים ברשימה מקום פנוי להכנסה:

| | |
|----------------------------------|--|
| StudentList () | הפעולה בונה רשימה כיתתית ריקה |
| void add (Student st) | הפעולה מוסיפה את התלמיד st לרשימה הכיתתית. |
| Student del (String name) | הפעולה מוחקת את התלמיד ששמו name מתוך הרשימה הכיתתית. הפעולה מחזירה את התלמיד שנמחק. אם התלמיד אינו קיים ברשימה הפעולה מחזירה null |
| Student getStudent (String name) | הפעולה מחזירה את התלמיד ששמו name. אם התלמיד אינו קיים ברשימה, הפעולה מחזירה null |
| String toString() | הפעולה מחזירה מחרוזת המתארת דף קשר כיתתי הממוין בסדר אלפביתי, באופן זה: <name1> <tel1> <name2> <tel2> |

רשימה כיתתית היא עצם המייצג אוסף של עצמים מטיפוס Student. לעצם כזה תהיה תכונה אחת או יותר לשמירת העצמים המייצגים את התלמידים, ולכן רשימה כיתתית היא עצם מורכב. הפעולות של StudentList, בנוסף לפעולות של המחלקות Student ו-Parent, מאפשרות למשתמש מרחב פעילות רחב.

? א. כתבו פעולה המקבלת רשימה כיתתית ושם של תלמיד, ומדפיסה את מספר הטלפון הסלולרי

של ההורה האחראי על תלמיד זה.

ב. יותם, הבכור באחי משפחת כהן, התייצב במזכירות בית הספר ודיווח על שינוי מספר

הטלפון הסלולרי של אביו. כיצד תבצע נטע המזכירה את העדכון? פרטו ונמקו בעזרת קוד

או בעזרת תרשימי עצמים.

1.1 שימוש בעצם מורכב

ניצור עצם מטיפוס StudentList ונמלא אותו בתלמידים.

לשם הנוחות נניח כי מאגר ההורים האחראיים כבר קיים וכי שלושה הורים נשלפו מתוכו והפניות

אליהם מצויות במשתנים: p1, p2 ו-p3. עוד נניח כי התלמידים הנרשמים בפעולה הנוכחית

מוכנסים כל אחד למאגר התלמידים ולרשימה הכיתתית:

```
public static void main(String[] args)
{
    // כאן מתבצעת שליפת שלושת ההורים מתוך המאגר

    StudentList studentList = new StudentList();
    Student st1 = new Student("Potter Harry", "02-9965187", p1);
    Student st2 = new Student("Scofield Michael", "03-6742201", p2);
    Student st3 = new Student("Messy Lionel", "08-9415632", p3);
    // כאן מתבצעת הכנסת שלושת התלמידים למאגר התלמידים

    // להלן הוספת שלושת התלמידים לרשימה הכיתתית:
    studentList.add(st1);
    studentList.add(st2);
    studentList.add(st3);

    System.out.println(studentList);
}
// ההדפסה, המחזירה דף קשר ממיון, תיראה כך:
```

Messy Lionel 08-9415632

Potter Harry 02-9965187

Scofield Michael 03-6742201

2.1 ייצוג ומימוש של עצם מורכב

כיוון שעצם של המחלקה מחזיק מידע על מספר רב של תלמידים, לא נכון להגדיר תכונה עבור כל

תלמיד. סביר יותר להגדיר תכונה מטיפוס מערך שתכיל את אוסף התלמידים. מספר התלמידים

הממוצע בכיתה בבית הספר הוא 30. לפי כללי בית הספר אין אפשרות שמספר התלמידים יעלה על 35, לכן נקבע שגודל המערך יהיה תמיד 35.

נראה את ייצוג המחלקה ואת הקוד המממש חלק מהפעולות:

```
public class StudentList
{
    public static final int MAX_STUDENTS = 35;
    private Student[] list;
    ...
}
```

הפעולה הבונה:

```
public StudentList ()
{
    this.list = new Student[MAX_STUDENTS];
}
```

פעולת ההוספה:

בפעולה הבונה של המחלקה StudentList, מוגדר מערך ללא אֶתחול. כיוון שזהו מערך של הפניות, המערכת מאתחלת כל תא במערך להפניה הריקה: null. כדי להתחיל שנת לימודים תקינה יש להכניס למערך את תלמידי הכיתה. הפעולה האחראית על ביצוע אתחול זה היא פעולת ההוספה, והיא אמורה להכניס לתא הראשון הפנוי במערך בכל שלב. כיצד נדע מהו מקום זה? ניתן לסרוק את המערך בעת ביצוע פעולת הכנסה כדי להגיע לתא הראשון שערכו null. זהו המקום המתאים להכנסה הנוכחית. מימוש יעיל יותר של פעולה זו אפשרי אם נגדיר תכונה נוספת, lastPosition, במחלקה StudentList. תכונה זו תשמור את האינדקס של המקום הפנוי הראשון. אינדקס זה יתעדכן בזמן ההוספה של תלמידים חדשים לרשימה הכיתתית. במימוש הנוכחי נפעל על פי גישה זו. בפעולה הבונה נאתחל את התכונה לאפס, ובפעולת ההוספה נכניס את התלמיד החדש למקום lastPosition:

```
public class StudentList
{
    public static final int MAX_STUDENTS = 35;
    private Student[] list;
    private int lastPosition;

    public StudentList()
    {
        this.list = new Student[MAX_STUDENTS];
        this.lastPosition = 0;
    }

    public void add(Student st)
    {
        this.list[this.lastPosition] = st;
        this.lastPosition++;
    }
}
```

? כתבו את פעולת ההוספה ללא הגדרת התכונה `lastPosition`. נתחו את זמן הריצה של פעולת ההוספה עם התכונה ובלעדיה, ונסחו את מסקנותיכם.

נחזור ונזכיר כי בעת זימון פעולת ההוספה מועבר אליה פרמטר שהוא הפניה לתלמיד החדש. הפרמטר בכותרת הפעולה הוא משתנה מקומי הנוצר ברגע הפעלת הפעולה, ומתבטל בסיומה.

? נניח כי הרשימה הכיתתית `lst` קיימת. במהלך תוכנית כלשהי, הוצבה במשתנה `student1` הפניה

לתלמיד ואז ביצעה התוכנית את הזימון הזה :

```
lst.add(student1);
```

א. התבוננו בפעולה זו :

```
this.list [this.lastPosition] = st;
```

כמה הפניות אל התלמיד החדש קיימות לפני ביצוע שורת קוד הבאה שבגוף פעולת ההוספה, וכמה קיימות אחריה. הסבירו את תשובתכם בפירוט.

ב. כמה הפניות לתלמיד יישארו עם תום פעולת ההוספה?

הגענו למסקנה שאליה הגענו כמה פעמים בפרק זה: לעצם אחד יכולות להיות הפניות שונות. פעילות עדכון ושינוי של העצם דרך הפניה אחת תשתקף גם כשנפנה אליו בעזרת ההפניות האחרות, ויש להיות מודעים לכך.

פעולת המחיקה:

הפעולות `getStudent()` ו-`del()`, המחזירות שתיהן הפניה אל תלמיד (או `null`), מצריכות סריקת המערך הפנימי של המחלקה, כדי למצוא אם התלמיד מופיע בו ולפעול בהתאם. כדי לחסוך השוואות ובדיקות מיותרות, סריקה זו תיעזר בתכונה `lastPosition` והיא תעדכן אותה בהתאמה (כחלק מן הפעולה `del(...)`). נזכיר כי השוואת מחרוזות נעשית באמצעות הפעולה `compareTo` (ראו ב-`javaAPI`).

לפניכם קוד הפעולה `del(...)`:

```
public Student del(String name)
{
    Student st = null;
    int i = 0;

    while (i < this.lastPosition &&
           this.list[i].getName().compareTo(name) != 0)
        i++;

    if (i < this.lastPosition)
    {
        st = this.list[i];
        // צמצום "חורים"
        for(int k = i+1; k < this.lastPosition; k++)
            this.list[k-1] = this.list[k];

        this.lastPosition--;
        this.list[this.lastPosition] = null;
    }
    return st;
}
```

הפעולה `toString()`:

על פי הגדרת פעולת ההוספה, כל תלמיד חדש מוכנס למקום הפנוי הראשון במערך. מכאן שהתלמידים מופיעים במערך לפי סדר הכנסתם. לעומת זאת, הפעולה `toString()` מחזירה מחרוזת המתארת את הרשימה באופן ממוין. אם כך, על הפעולה למיין את רשימת התלמידים לפני החזרת דף הקשר.

פעולת המייון היא פעולה יקרה אם מדברים במושגים של יעילות.

האם יש אפשרות להימנע ממייון הרשימה בביצוע הפעולה `toString()`?

אם נוודא שהמערך המייצג את הרשימה הכיתתית נשמר ממוין כל הזמן, לא נצטרך להשקיע משאבים במייון לקראת הפקת דף הקשר. ואולם, לשם כך, יהיה עלינו לממש אחרת את פעולת ההוספה והמחיקה, כך שכל תלמיד יוכנס אל המערך או יוצא ממנו על פי מקומו הנכון בסדר אלפביתי.

בסעיף הבא נבחן מימוש חילופי שכזה.

3.1. מימוש חילופי

האם הייצוג של המחלקה משתנה בעקבות ההחלטה על שינוי המימוש? לכאורה ניתן היה לוותר על הגדרת התכונה `lastPosition`. בכל זאת, עדיף להשאיר את התכונה ולהיעזר בה. נשאיר את הייצוג כשהיה ונשנה רק את מימוש הפעולות הבאות.

פעולת ההוספה

נכתוב מחדש את פעולת ההוספה, כך שהמערך יישמר ממוין כל הזמן. המערך הראשוני הנוצר עבור הרשימה אינו מכיל תלמידים, ולכן הוא ממוין. עלינו לוודא שכל תלמיד שיוכנס אליו, יוכנס ישירות למקומו הנכון על פי סדר אלפביתי. כך יישמר המערך ממוין כל זמן.

האלגוריתם של פעולת ההוספה השומרת על המיון של המערך ייראה כך :

1. סרוק את המערך מתחילתו. כל עוד שם התלמיד החדש גדול (לפי סדר אלפביתי) משם התלמיד במקום הנוכחי – התקדם למקום הבא במערך.
2. הזז ימינה (מקום אחד) את כל איברי המערך מהמקום הנוכחי עד סוף המערך (כדי ליצור מקום במערך שלתוכו ניתן יהיה להכניס את התלמיד החדש).
3. הכנס את התלמיד החדש למקום שהתפנה.

בשלב הראשון של האלגוריתם אנו מוצאים את המקום שאליו צריך להכניס את התלמיד החדש. המקום להכנסה הוא זה שבו הסריקה נעצרת. הנחת המוצא של היישום הבית ספרי הייתה שאין שמות כפולים בעולם. תנאי הסריקה הוא שבנקודת העצירה שמות התלמידים הקודמים קטנים משם התלמיד החדש, והשם במקום הנוכחי כבר גדול ממנו. לכן המקום שהתפנה הוא המקום שאליו יש להכניס את התלמיד החדש. כדי לשמור על הסדר יש להזיז את כל התלמידים ממקום זה והלאה מקום אחד ימינה. ההנחה המקדימה הנוספת ביישום אומרת שתמיד יהיה מקום במערך להכנסת תלמיד חדש ואין צורך לבדוק זאת.

? מה יקרה במקרי הקצה בפעולת ההוספה, למשל כאשר המקום הנכון להכנסה הוא המקום הראשון או המקום האחרון במערך?

הקוד שלפניכם מממש את האלגוריתם המתואר תוך שהוא משתמש בפעולת העזר `moveOthers(...)`, המממשת את שלב 2 באלגוריתם. במקרה הקצה – כשהוספת התלמיד נעשית בסוף המערך – אין צורך להפעיל את הפעולה :

```
public void add(Student student)
{
    int i = 0;

    while (i < this.lastPosition &&
           this.list[i].getName().compareTo(student.getName()) < 0)
        i++;

    if (i < this.lastPosition)
        moveOthers(i);

    this.list[i] = student;
    this.lastPosition++;
}
```

? א. ממשו את הפעולה `void moveOthers (int place)`.

ב. מכיוון שהמערך ממזין עולה השאלה האם ניתן להיעזר בחיפוש בינרי ולהקטין את סדר הגודל של פעולת ההוצאה ל- $O(\log n)$? הסבירו ונמקו את תשובתכם.

4.1. יעילות הפעולות בשתי דרכי המימוש

ראינו שתי דרכים שונות למימוש המחלקה המאפשרות לנו להדפיס את דף הקשר של התלמידים ממזין בסדר אלפביתי. השינוי בדרך המימוש משפיע על היעילות. ננתח את הפעולות בהתאם למימוש:

הדרך הראשונה היא למזין את המערך רק בעת זימון הפעולה `toString()`. בדרך זו, ההוספה היא פשוטה. האיבר מוכנס למקום האחרון במערך, ומקום זה נשמר כתכונה במחלקה. סדר הגודל של יעילות פעולת ההוספה יהיה קבוע – $O(1)$. לעומת זאת, יעילות הפעולה `toString()`, הכוללת את מיון המערך, אם ניעזר בפעולה מיון מיזוג שהכרנו בפרק יעילות, תהיה $O(n \log n)$.

הדרך האחרת היא לשמור את המערך ממזין מרגע ההכנסה של האיבר הראשון. זאת נעשה על ידי הכנסת האיברים בזה אחר זה למקום הנכון לפי הסדר האלפביתי. בדרך זו יעילות ההוספה היא $O(n)$, כיוון שנדרשת סריקת איברי המערך כדי למצוא את המקום הנכון להכנסה, וכיוון שהאיברים הנמצאים ממקום זה והלאה מוזזים מקום אחד ימינה.

יעילות הפעולה `toString()` עצמה תהיה $O(n)$, מכיוון שהמערך כבר ממזין, ויש רק לסרוק אותו לצורך ההדפסה.

הבחירה בין שתי החלופות תלויה במספר הפעמים הצפוי לביצוע פעולות ההוספה והדפסת דף הקשר. ביישום הנוכחי, סביר שכל תלמיד יוכנס פעם אחת, אך דף הקשר יודפס פעמים רבות במהלך שנת הלימודים, ולכן החלופה השנייה המחזיקה את הרשימה הכיתתית ממזינה כל הזמן עדיפה.

5.1. כמה הערות לסיכום

בפרק זה הצגנו (לראשונה בספר זה) יישום "כמעט אמיתי" – יישום של מערכת לניהול בית ספרי. המערכת משתמשת בכמה נתונים: בעצמים מהטיפוסים `Parent` ו-`Student`; באוספים של עצמים אלה, כלומר במאגרי ההורים והתלמידים (שבמימושם לא דנו); וברשימות כיתתיות מטיפוס `StudentList`.

הטיפוס `Student` מגדיר עצמים מורכבים, כיוון שכל עצם מטיפוס "תלמיד" מכיל הפניה לעצם מטיפוס "הורה אחראי". האוספים אף הם עצמים מורכבים, המכילים הפניות לעצמים רבים אחרים. לא כאן המקום המתאים לדון בפירוט כיצד צריך יישום מורכב כזה להיות מאורגן, ואנו נסתפק בשתי הערות.

הערה 1: ברשימה כיתתית אנו משתמשים בשם כמזהה של התלמיד, ומניחים שאין שני תלמידים בעלי אותו השם. סביר ששם התלמיד מזהה אותו גם במאגר התלמידים הכללי, ובאופן דומה שם

ההורה מזהה הורה במאגר ההורים (למעשה, ביישום אמיתי, הנחה זו אינה תקפה, כיוון שייכתנו שני תלמידים ויותר בעלי אותו השם. לכן מקובל להשתמש במספר תעודת הזיהוי כמזהה. לשם הפשטות והנוחות השתמשנו כאן בשמות).

עד עכשיו, בדרך כלל, כאשר הגדרנו מחלקות חדשות, התכוונת שבהן היו פרטיות, ולכל תכונה הגדרנו פעולות קריאה ושינוי – `get()`, `set(...)`. אולם, אין חובה לספק פעולות אלה לכל תכונה. בפרט, כאשר עצם, כגון `Student` מכיל תכונה המיועדת לשמש כמזהה, אין לאפשר שינוי שלה. שינוי שם של תלמיד יכול לגרום בעיות. ראשית, אי אפשר יותר לסמוך על ההנחה שאין ברשימה כיתתית שני תלמידים בעלי אותו השם. ייתכן שההנחה נכונה כאשר נוצרת הרשימה, אך בזמן כלשהו מאוחר יותר, אם שונה שמו של אחד התלמידים, ההנחה אינה ודאית עוד. אם השינוי מתבצע בחלק אחר של התוכנה, שנכתבה על ידי מתכנת אחר, ועל ידי משתמש שאינו קשור לכיתה, ייתכן שהעובדה הזו לא תילקח בחשבון. יתר על כן, גם אם השם החדש שונה מכל שמות התלמידים האחרים בכיתה, ובחרנו לאחסן רשימה כיתתית במערך ממוין לפי שמות, הרי לאחר השינוי קרוב לוודאי ששם התלמיד שוב אינו נמצא במקום המתאים לו במערך לפי המיון.

הערה 2: כבר הזכרנו פעמים מספר כי ייתכן שיהיו כמה הפניות לאותו העצם. ביישום שאנו דנים בו, בעצם "תלמיד" יש הפניה לעצם "הורה". במאגר ההורים יש הפניות להורים, ובמאגר התלמידים וכן ברשימות הכיתתיות יש הפניות לתלמידים. מצב כזה אופייני וטבעי ליישום כזה. ראינו קודם שניתן לשנות עצם מכל הפניה אליו. ביישום הזה, ראינו כי מורה או מזכירה המגיעה לעצם של הורה על ידי שימוש בהפניה אליו שנמצאת בעצם של תלמיד, יכולה לשנות את מספר הטלפון של ההורה. העובדה ששינוי זה ישתקף בעצמים של כל האחים של התלמיד (שמפנים לאותו הורה) היא טובה ומתאימה ליישום. אולם, קיום של כמה הפניות לאותו העצם הוא גם פתח לסיבוכים ולשגיאות לוגיות, משום שהנגישות לעדכונן קיימת דרך כל אחת מההפניות. אם, בגלל תכנון שגוי, היינו מאפשרים עדכון שם של תלמיד, אזי עדכוננו במקום כלשהו במערכת עלול לפגוע בתכונות החיוניות לפעולה תקינה של הרשימה הכיתתית, כלומר לפגוע בעצם הקיים בחלק אחר ומנותק של התוכנית. המסקנה היא שיש לתכנן בקפדנות את מספר ההפניות לכל עצם ולשלט בהן. כמו כן יש לוודא ששינוי המתבצע דרך כל אחת מההפניות לא יפגע בתקינות העצמים המכילים את ההפניות האחרות.

נסיים את הדיון בהפניות בהצגת בעיה אפשרית אחרת. כאשר כיתה יוצאת לטיול אנו מעוניינים לתת לאחראי הטיול גישה לרשימת התלמידים העתידיים לצאת לטיול. הבה נניח מצב דמיוני מעט (אך סביר מאוד בעוד כמה שנים), שניתן להתקשר למערכת הבית ספרית מכל מקום בארץ, ולבצע בה פעולות. נקרא לרכיב החדש "רשימת טיול". לכאורה כל שעלינו לעשות הוא לאחסן ביישום החדש הפניה לרשימה הכיתתית. קל לראות כי דבר זה עלול להוליד בעיות. אם מסיבה כלשהי נבצר מילד כלשהו להשתתף בטיול, אחראי הטיול ימחק אותו מרשימת הטיול. מה ניתן לומר על הרשימה הכיתתית אחרי שהטיול הסתיים?

המסקנה היא שכאשר אנו מעוניינים להשתמש ברשימה הכיתתית לצורך חדש, שבו יש אפשרות לפעולות של הוספה או מחיקה של תלמידים, אסור לנו לעשות זאת על ידי שימוש בהפניה לרשימה עצמה. הדבר הנכון לעשות הוא ליצור עותק של הרשימה, שיינתן לאחראי הטיול. הסעיף הבא יעסוק בהענקות כאלה.

ז. העתקה

בפרק הדין במחלקות הסברנו כי כאשר יש צורך לייצר עותק של עצם, השונה ממנו בזהותו, אך שווה לו בערכי תכונותיו, יש להשתמש בפעולה בונה מעתיקה. כאשר מדובר בעצם מורכב a, שאחת מתכונותיו היא הפניה לעצם אחר b, עלינו לבחור בין שתי אפשרויות:

- i. העתקת ההפניה ל-b לעצם החדש. כך יהיו שתי הפניות לאותו עצם b.
- ii. העתקת העצם b לעצם חדש c, ואחסון ההפניה לעצם החדש c, בעותק של a.

נחזור לדוגמה של הרשימה הכיתתית ורשימת הטיול. עומדות לפנינו שתי אפשרויות למניעת הפגיעה ברשימה. האחת לייצר עותק של הרשימה הכיתתית. לעותק זה יועתקו ההפניות המקוריות שהיו ברשימה הכיתתית, ולכן הוא יכיל הפניות אל העצמים של התלמידים עצמם. האפשרות האחרת היא ליצור עותק של הרשימה הכיתתית, שיכיל עותקים של עצמי התלמידים.

נדגים את האפשרות הראשונה ליצירת עותק של רשימה כיתתית, כפעולה בונה מעתיקה של המחלקה StudentList. הפעולה תיצור עצם חדש מטיפוס רשימה כיתתית אך הוא יכיל הפניות אל מופעי התלמידים המקוריים עצמם, שהרי ההפניות המקוריות יוצבו לתוכו:

```
public StudentList(StudentList stList)
{
    this.list = new Student[MAX_STUDENTS];

    for (int i=0; i<stList.lastPosition; i++)
        this.list[i] = stList.list[i];

    this.lastPosition = stList.lastPosition;
}
```

נבצע כעת את הזימון הבא לפעולה הבונה המעתיקה:

```
StudentList tripList = new StudentList(studentList);
```

אמנם אחראי הטיול מחזיק עכשיו עותק של הרשימה הכיתתית, אך יש לו עדיין גישה ישירה למופעי התלמידים. לפיכך, אם הוא ישנה את פרטיו של תלמיד, ישתנו פרטיו של התלמיד במאגר התלמידים. מצב זה נראה לא תקין. אחראי הטיול אינו מוסמך לשנות פרטים אלה (שימו לב שלפי חלוקת הסמכויות בבית הספר סביר שמחנך הכיתה יהיה רשאי לבצע שינויים שכאלה).

אם כך, הבה נבחן אפשרות אחרת. נניח שהפעולה הבונה המעתיקה של המחלקה StudentList תיצור עותק של הרשימה המקורית שבו יופיעו עותקים של מופעי התלמידים. עותקים אלה ייווצרו על ידי פעולה בונה מעתיקה שנוסיף למחלקה Student:

```
public StudentList(StudentList stList)
{
    this.list = new Student[MAX_STUDENTS];

    for (int i=0; i<stList.lastPosition; i++)
        this.list[i] = new Student(stList.list[i]);

    this.lastPosition = stList.lastPosition;
}
```

? כמוכן שאם איננו רוצים שאחראי הטיול יעדכן את פרטי התלמידים, אנו גם לא מעוניינים לאפשר לו לעדכן את פרטי ההורים (דרך עצמי התלמידים). כיצד נמנע זאת?

הערה: ג'אוה מרשה העמסה של פעולות בונות, אך רק אם הכותרות שלהן שונות (ברשימת הפרמטרים של הפעולות). שתי הפעולות הבונות שהצגנו הן בעלות אותה רשימת פרמטרים, ולכן נוכל לכלול במחלקה רק אחת מהן, או שנצטרך לשנות את הכותרת של אחת מהן לפחות.

לסיכום: ראינו מקרים שונים של שימוש בכמה הפניות לאותו העצם, וכן גישות להימנע מבעיות שיכולות להתעורר כאשר יש כמה הפניות לאותו העצם.

אין כללים חד-משמעיים המורים לנו מתי כדאי להעתיק עצם ומתי רצוי להשתמש במקור. הדבר תלוי בהגדרת הבעיה. בכל מקרה לגופו, צריך לקחת בחשבון את ההשלכות של שימוש בעצם או בעותק.

ח. עצם שתכונתו מטיפוס עצמו

ראינו כי עצם מורכב הוא עצם שלו תכונה מטיפוס של מחלקה כלשהי. בפרט אנו יכולים לומר שתכונתו של העצם יכולה להיות מטיפוס המחלקה שלו עצמו.

נתבונן לדוגמה במחלקה Bead המגדירה חרוז. חרוז הוא עצם מורכב. לעצם מטיפוס המחלקה Bead שתי תכונות: צבע והפניה לחרוז המושחל אחריו בשרשרת. שימו לב, כי התכונה השנייה היא מטיפוס המחלקה שבה תכונה זו מוגדרת, כלומר היא מגדירה הפניה לעצם מטיפוס Bead. מחרוזים כאלה נוכל להכין שרשרת של חרוזים, כלומר לשרשר את החרוזים זה לזה באמצעות ההפניה שהם מכילים.

| Bead |
|---|
| String color Bead nextBead |
| Bead (String color) Bead (String color, Bead nextBead) String getColor() Bead getNextBead() void setNextBead (Bead nextBead) |

פעולות הממשק אינן כוללות פעולה לשינוי צבעו של חרוז. מרגע היצירה, צבעו של חרוז אינו ניתן לשינוי.

נראה את מימוש הפעולות `set(...)` ו-`getNextBead()` המתייחסות לערך התכונה `nextBead`:

```
public Bead getNextBead()
{
    return this.nextBead;
}

public void setNextBead(Bead nextBead)
{
    this.nextBead = nextBead;
}
```

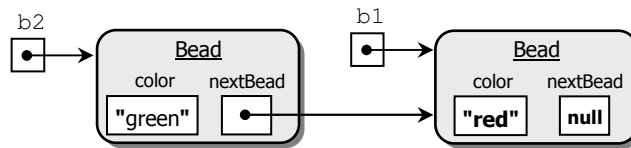
בעזרת פעולות אלה ניתן לשרשר חרוזים ולייצר שרשרת. ניצור שני חרוזים בודדים בעזרת הפעולה הבונה המתאימה:

```
Bead b1 = new Bead("red");
Bead b2 = new Bead("green");
```



```
b2.setNextBead(b1);
```

עתה נחרוז את החרוזים לשרשרת:



דרך שונה ליצירת שרשרת כזו היא תוך שימוש בפעולה הבונה באופן הזה:

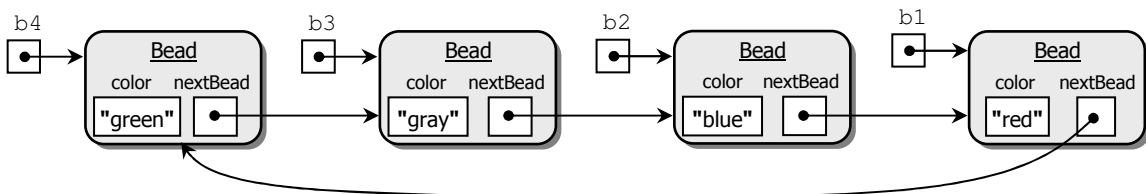
```
Bead b1 = new Bead("red");
Bead b2 = new Bead("green", b1);
```

שימו לב שלאור כל הנאמר בפרק עד עתה, ברור כי ככל שקיימות יותר הפניות לחרוזים בשרשרת, קל יותר לקלקל את המבנה שלה. חריזה שבה לא יישארו הפניות חיצוניות תאפשר בנייה של שרשרת בטוחה ועמידה יותר. לפניכם דרך נוספת לחריזת אותם שני החרוזים:

```
Bead b2 = new Bead("green", new Bead("red"));
```

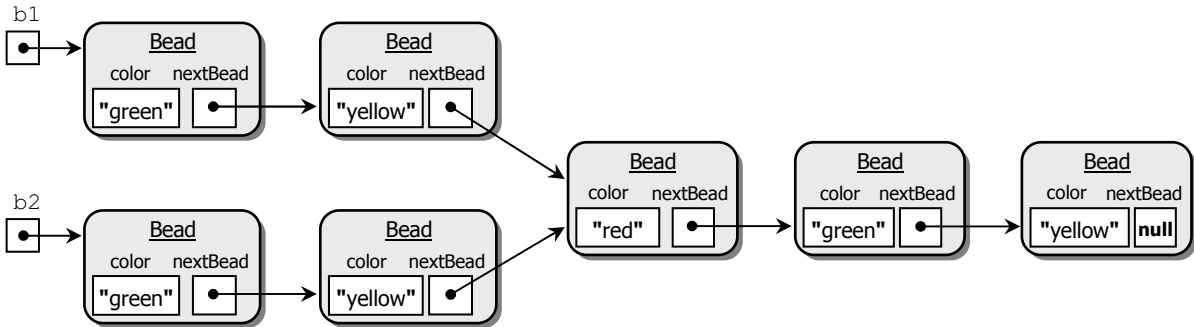
בעזרת המחלקה "חרוז" ופעולותיה אנו יכולים ליצור שרשרות שונות. ראשית ניצור שרשרת קלסית של חרוזים, כלומר מעגל סגור של חרוזים:

```
Bead b1 = new Bead("red");
Bead b2 = new Bead("blue", b1);
Bead b3 = new Bead("gray", b2);
Bead b4 = new Bead("green", b3);
b1.setNextBead(b4); // סגרנו את השרשרת
```



? כתבו קטעי קוד עבור שרשרות החרוזים המתוארות להלן:

- א. הרכיבו שתי שרשרות שונות, הזחות הן בצבעי החרוזים והן בסדר הופעת הצבעים בשרשרת. כל שרשרת תהיה מורכבת מחמישה חרוזים, שצבעיהם (משמאל לימין) הם: ירוק, צהוב, אדום, ירוק, צהוב. המשתנים b1 ו-b2 יכילו הפניות לחרוזים הראשונים שבשתי השרשרות.
- ב. חברו את שתי השרשרות שיצרתם על ידי שינוי הפניה בשרשרת השנייה, כך שיתקבל מבנה כמתואר באיור הבא. (שימו לב שאחרי ביצוע המשימה, אין בתוכניתכם אף הפניה לשלושת החרוזים האחרונים שבשרשרת השנייה).



- ג. בנו מחדש את השרשרות המקוריות שבסעיף א. הרכיבו שרשרת ארוכה המורכבת משתי השרשרות המקוריות, תחילה הראשונה ואחריה השנייה, וסגרו אותה למעגל.

1.ח. שרשור היררכי של עצמים מאותו הטיפוס

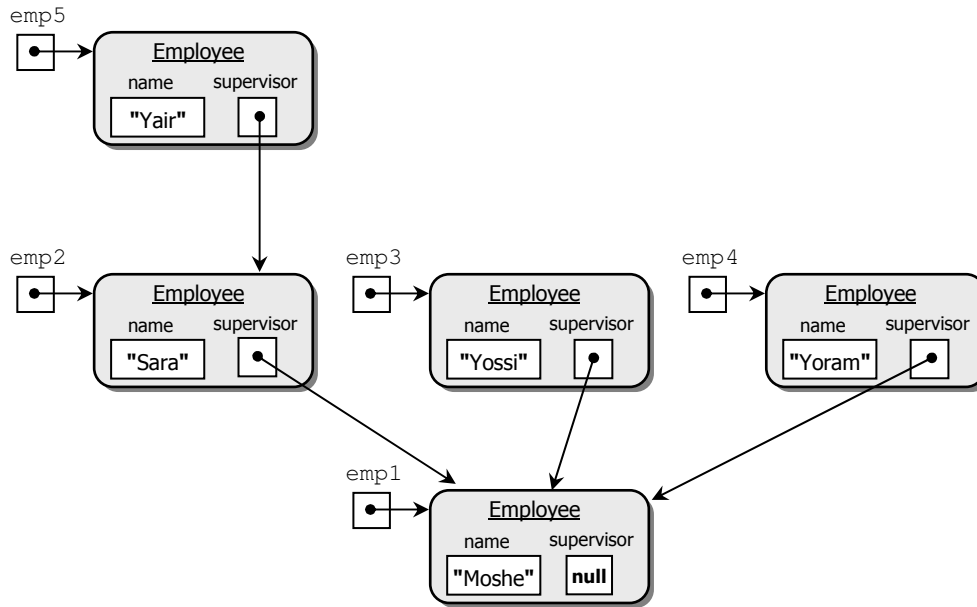
החרוזים המשורשרים שראינו בסעיף הקודם יצרו מבנה לינארי ישר או סגור. השרשרות שהתלכדו לשרשרת אחת מאמצען הציגו מבנה לינארי מורכב מעט יותר. נוכל לנצל את המבניות הזו כדי לייצג מבנים שבהם השרשור מבטא סדר כלשהו. בדוגמה הבאה יציג המבנה המשורשר של העצמים היררכיה בין איבריו.

המחלקה Employee מגדירה עובד בחברה. לכל עובד בחברה, למעט מנהל החברה, יש אחראי שגם הוא עובד בחברה. אחת התכונות של המחלקה Employee, תהיה אם כן, מטיפוס Employee, ובה תישמר הפנייה לאחראי של העובד. אם לעובד אין אחראי תכיל התכונה, הפניה ריקה:

| Employee |
|--|
| String name |
| Employee supervisor |
| Employee(String name, Employee supervisor) |
| String getName() |
| Employee getSupervisor() |
| void setSupervisor (Employee supervisor) |

היישום שלפניכם יוצר מבנה משורשר של עצמים מטיפוס Employee, המייצג מבנה ארגוני קלסי:

```
public static void main(String[] args)
{
    Employee emp1 = new Employee("Moshe", null);
    Employee emp2 = new Employee("Sara", emp1);
    Employee emp3 = new Employee("Yossi", emp1);
    Employee emp4 = new Employee("Yoram", emp1);
    Employee emp5 = new Employee("Yair", emp2);
}
```



לפני שנסיים את הפרק ברצוננו לדון בעוד שני נושאים שעניינם הפניות.

ט. מחרוזת – עצם מקובע

בג'אווה מחרוזות הן עצמים שאינם ניתנים לשינוי – עצמים מקובעים (immutable). כאשר אנו "משנים" מחרוזת, בפועל נוצרת תמיד מחרוזת חדשה, אך המחרוזת המקורית אינה משתנה. למשל, נניח שמוגדרת המחרוזת:

```
String str1 = "Moshe";
```

כדי שנוכל לבחון את ההתנהגות של המחרוזות בעת ביצוע השינוי, נעתיק אותה (כיוון שמחרוזות היא עצם, זו למעשה העתקה של הפניה למחרוזת) למשתנה נוסף:

```
String str2 = str1;
```

נרשר ל-str1 מחרוזת נוספת, למשל:

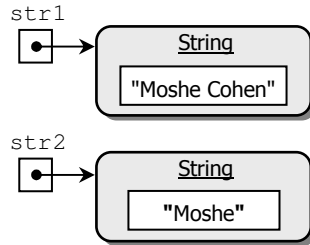
```
str1 += " Cohen";
```

למדנו שכאשר משנים עצם שיש אליו שתי הפניות, השינוי נראה דרך שתיהן. אבל, בפועל נוצרת מחרוזת חדשה שערכה הוא "Moshe Cohen". ההפניה במשתנה str1 תפנה עכשיו לשם זה, אך המחרוזת "Moshe" המקורית לא תשתנה. נוכל להיווכח באמיתות הטענה על ידי הדפסת שני המשתנים:

```
System.out.println(str1 + ", " + str2);
```

הפלט שיתקבל יהיה: Moshe Cohen, Moshe. החלק הראשון הוא המחרוזת החדשה ש-str1 מפנה אליה. החלק השני הוא המחרוזת שהמשתנה str2 מפנה אליה, וקל להיווכח שזוהי אכן המחרוזת המקורית.

לפניכם מוצג מצב הזיכרון בעזרת תרשים עצמים:



תכונה זו של מחרוזות חשובה מאוד לדיוננו בפרק זה, משום שעצמים רבים מכילים מחרוזות, ופעולות שלהם מחזירות מחרוזות, למשל, הפעולה getName() של עצמים מטיפוס Student. אחרי שימוש בפעולה כזו, ייתכן שכמה משתנים מחזיקים הפניה לאותה המחרוזת; אחד מהם הוא התכונה של עצם Student והאחר הוא משתנה בפעולה חיצונית. למרות זאת, אין בזה סכנה לתקינות הנתונים, כיוון שמשמש המקבל שם כערך החזרה אינו יכול לשנותו. אוספים כגון רשימה כיתתית או ספר טלפון, משתמשים בשמות, כלומר במחרוזות, כמזהים. אילו הייתה אפשרות לכל מי שמחזיק הפניה לשם לשנות אותו, הייתה זו סכנה לתקינות מבנה האוסף. העובדה שמחרוזות אינן ניתנות לשינוי מונעת את הסכנות הללו. סביר שזו הסיבה העיקרית לכך שמחרוזות אינן בנות שינוי.

התנהגות המחרוזות למעשה זהה להתנהגותם של ערכים מטיפוסים בסיסיים. למשל, נניח שמשתנה num1 מכיל את המספר 7, והוא מועתק למשתנה num2 ואחר כך מבצעים:

```
num1 += 3;
```

קעת המשתנה num1 מכיל את המספר 10, אך num2 עדיין מכיל את המספר 7. לכן, אף שמחרוזות הן עצמים, ניתן להתייחס לעצם מטיפוס מחרוזת במידה רבה כאילו הוא ערך מטיפוס בסיסי.

י. אספן זבל

בג'אווה קיים מנגנון מיוחד בשם **אספן זבל (garbage collector)**, הדואג למחוק מהזיכרון עצמים שאין אליהם הפניות ממשתנים של התוכנית. מכיוון שבמקרים כאלה לא ניתן לגשת אל העצמים הללו הם הופכים מיותרים והמנגנון מפנה את תאי הזיכרון שהם תופסים.

לדוגמה:

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

אם לאחר ההקצאה:

```
b1 = null;
```

מציבים:

ההפניה היחידה מהתוכנית אל המופע של הדלי נמחקה ומנגנון איסוף הזבל ישחרר את העצם b1 מהזיכרון.

יא. סיכום

- "עצם מאוחסן במשתנה" פירושו שבמשתנה יש הפניה לעצם.
- אם מציבים במשתנה אחד הפנייה הנמצאת במשתנה אחר, שני המשתנים "יכילו" למעשה אותו העצם.
- השוואת הפניות פירושה בדיקה האם ההפניות שוות, כלומר האם הן הפניות לאותו העצם, ולא בדיקה האם העצמים שווים בערכי התכונות שלהם.
- פנייה למשתנה מטיפוס מחלקה שאינו מאותחל תגרום לשגיאת הידור.
- ניתן להציב במשתנה מטיפוס מחלקה את הערך `null`, מה שאומר שהמשתנה אינו מכיל הפניה לעצם. ניסיון לזמן פעולות של עצם על ידי פנייה למשתנה המכיל `null` יגרור שגיאת ריצה: `NullPointerException`.
- עצם מורכב הוא עצם שלפחות אחת מתכונותיו היא עצם.
- אם מציבים הפניה חיצונית לתוך תכונה של עצם מורכב, אזי הן ההפניה החיצונית והן התכונה של העצם פונות לאותו העצם. שינוי שיעשה דרך אחת ההפניות ישנה את העצם, ולכן כאשר ניגש אל תכונות העצם דרך ההפניה השנייה נרגיש בשינוי.
- כאשר מפעילים פעולה המאחזרת את ערכה של תכונה שהיא עצם, ניתן לשמור את ההפניה הזו על ידי משתנה חיצוני. שוב נוצר מצב שבו הן ההפניה החיצונית והן התכונה של העצם פונות לאותו העצם. שינוי שיעשה דרך אחת ההפניות ישנה את העצם. לכן כאשר ניגש דרך ההפניה השנייה אל העצם נרגיש בשינוי.
- כדי ששתי הפניות יפנו לעצמים שונים בעלי תכונות זהות, ניתן להשתמש בפעולה בונה מעתיקה.
- בכל בעיה יש לשקול האם רצוי להשתמש בעותק של עצם או בעצם עצמו, בהתאם לתנאי הבעיה.
- ניתן להגדיר מחלקה שהתכונה שלה היא מטיפוס המחלקה עצמה.
- עצמים שלהם תכונה שהיא עצם מטיפוס עצמם, ניתנים לשרשור.
- מחרוזת היא עצם מקובע, אשר לא ניתן לשינוי. כל שינוי במחרוזת נעשה על ידי יצירת עצם חדש ובו השינוי הרצוי. תכונה זו של מחרוזת מונעת מצב שבו יהיו בזיכרון שתי הפניות לאותו העצם, ושינוי העצם באמצעות הפניה אחת ישפיע על העצם המוחזק על ידי ההפניה האחרת.
- אספן הזבל הוא מנגנון של שפת ג'אווה האחראי על שחרור שטחי זיכרון שאין אליהם הפניה חיצונית מתוך תוכניות.

מושגים

| | |
|-------------------|----------------|
| garbage collector | אספן הזבל |
| reference | הפניה |
| immutable | לא ניתן לשינוי |
| composite object | עצם מורכב |

פרק 6 דף עבודה מס' 1

הצבה של הפניות לעומת העתקה

מה עליכם לעשות:

1. הוסיפו פעולה בונה מעתיקה (copy constructor) למחלקה Point שהגדרתם בדף עבודה מס' 1 של פרק 3.
2. לפניכם פעולה ראשית:

```
public static void main(String[] args)
{
    Point p1 = new Point(4, 9);
    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 תרשים עצמים אחר ביצוע התוכנית וציינו מהו הפלט שיתקבל. הריצו את התוכנית, ובדקו את תשובותיכם.
2. כמה עצמים מטיפוס Point נוצרו בתוכנית הראשית? הסבירו.

בהצלחה!

פרק 6 דף עבודה מס' 2 ניקח נקודות ונבנה מלבן

המחלקה Rectangle

המחלקה Rectangle מגדירה מלבן שצלעותיו מקבילות לצירים.

| | |
|--|---|
| Rectangle (Point bottomLeft, Point topRight) | הפעולה בונה מלבן ומאתחלת את תכונותיו על פי הפרמטרים הנתונים: הנקודה השמאלית התחתונה, הנקודה הימנית העליונה |
| Rectangle (Point bottomLeft, double width, double height) | הפעולה בונה מלבן ומאתחלת את תכונותיו על פי הפרמטרים הנתונים: הנקודה השמאלית התחתונה, רוחב המלבן, גובה המלבן |
| double getArea() | הפעולה מחזירה את השטח של המלבן |
| double getPerimeter() | הפעולה מחזירה את ההיקף של המלבן |
| void move (double deltaX, double deltaY) | הפעולה מזיזה את המלבן: <ul style="list-style-type: none"> 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 |
| String toString() | הפעולה מחזירה מחרוזת המתארת את המלבן באופן הזה: Rectangle: bottom-left point = (<X> , <Y>) top-right point = (<X> , <Y>) |

שימוש בפעולות המלבן לדוגמה:

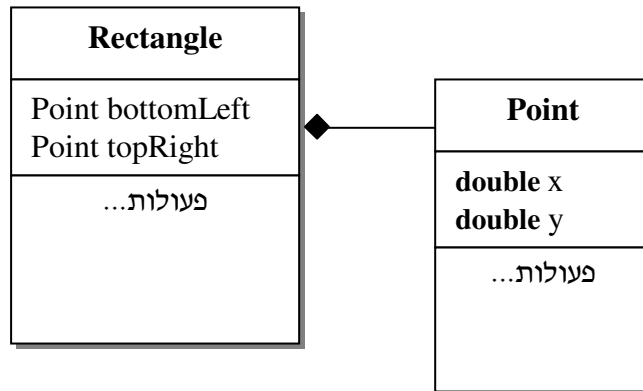
ניצור מלבן על פי הנתונים של שני הקודקודים המגדירים אותו. הערה: יש ליצור מלבנים שלהם נתונים תקינים (אין צורך לבצע בדיקות תקינות).

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

לאחר הפעלת הפעולה: move (-1, 2), כל הקודקודים ישתנו באותו היחס, 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 הוא 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.

מה עליכם לעשות?

1. כתבו את המחלקה Rectangle, בעלת הממשק המתואר לעיל, על פי הייצוג של שתי הנקודות. אל תשכחו לתעד את המחלקה באופן מלא.
2. כתבו תוכנית בדיקה קצרה שתוכיח את תקינות המחלקה Rectangle.
3. שנו את ייצוג המחלקה Rectangle. אילו תכונות בחרתם למחלקה כעת?
4. האם שינוי הייצוג של המחלקה יגרום לשינוי בממשק שלה? היכן ייעשו השינויים הנדרשים בעקבות שינוי הייצוג? פרטו שלושה שינויים שיש לבצע במימוש הפעולות.
5. האם יש צורך להשתמש בפעולה בונה מעתיקה ביצירת המלבן? אם כן, הסבירו היכן ומדוע?

בהצלחה!

פרק 6 דף עבודה מס' 3 נוסע ודרכון

המחלקה Passport

המחלקה Passport מגדירה דרכון של נוסע. לשם מימושה, עליכם להשתמש במחלקה Date.

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

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 שמוגדרים באיור הבא, למחלקה Traveler תכונה שהיא מופע של המחלקה Passport.

| | |
|--|---|
| Traveler (Passport passport, boolean hasPaid) | הפעולה בונה נוסע חדש ומאתחלת את תכונותיו על פי הנתונים |
| void pay() | הפעולה מבצעת תשלום של הנוסע עבור הנסיעה |
| boolean hasPaid() | הפעולה מחזירה true אם ורק אם הנוסע שילם עבור הנסיעה |
| String toString() | הפעולה מחזירה מחרוזת המתארת את הנוסע (על פי המבנה המתואר למטה) |
| 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 |

המחרוזת המתארת את הנוסע צריכה להיראות כך :

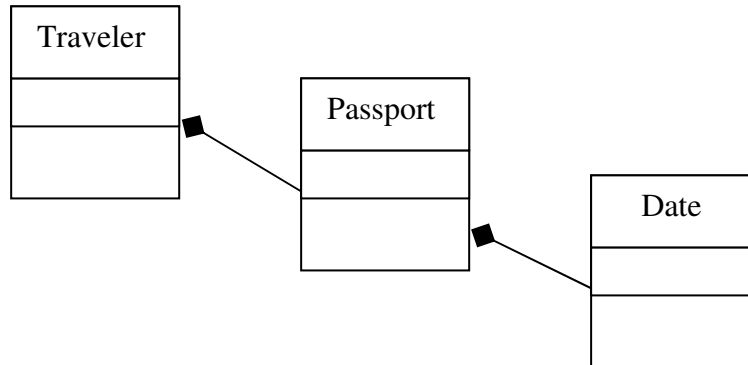
Name: <name>

Pass. num: <number>

Exp date: <expiry date>

Has paid: <hasPaid>

היחסים והקשרים בין המחלקות מתוארים באיור שלפניכם :



מה עליכם לעשות?

בדף עבודה זה עליכם לכתוב את המחלקות הבאות :

Passport – מגדירה דרכון של נוסע.

Traveler – מגדירה נוסע.

כדי לכתוב את המחלקות השתמשו במחלקה Date שכתבתם בפרק 3. עליכם להוסיף לה פעולה

בונה מעתיקה (copy constructor).

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 2008.

בהצלחה!

פרק 6 דף עבודה מס' 4 עצם מורכב

מה עליכם לעשות?

1. השלימו את כותרות הפעולות של המחלקה A :

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

2. כתבו את המחלקה A.

3. מהן התכונות החייבות להופיע במחלקה A? נמקו את תשובותיכם.

4. מהו מספר הפעולות הבונות שבמחלקה? כיצד נקרא המנגנון שמאפשר הימצאות של יותר מפעולה בונה אחת במחלקה?

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;
    }
}
```

בתוכנית הראשית קיימת פעולה זו:

```
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 המקום בזיכרון.

בהצלחה!

פרק 6 דף עבודה מס' 5 ספר טלפונים

ספר טלפונים (Phone Book) הוא אוסף של אנשי קשר (Contacts) שאינו מוגבל בגודלו. לכל איש קשר יש שם מלא ומספר טלפון. שם איש הקשר משמש כמזהה, לכן בספר הטלפונים לא יכולים להיות שני אנשי קשר (או יותר) בעלי שם זהה.

המחלקה PhoneBook

המחלקה PhoneBook מגדירה ספר טלפונים.

| | |
|--|---|
| PhoneBook() | הפעולה בונה ספר טלפונים ריק |
| void addContact (String name, String phone) | הפעולה מוסיפה לספר הטלפונים איש קשר ששמו name ומספר הטלפון שלו phone. אם איש קשר בשם זה קיים בספר הטלפונים, הפעולה מעדכנת את מספר הטלפון שלו לערך של phone |
| void delContact (String name) | הפעולה מוחקת מספר הטלפונים איש קשר ששמו name. אם איש הקשר אינו קיים, הפעולה אינה מבצעת דבר |
| String getPhone (String name) | הפעולה מחזירה את מספר הטלפון של איש הקשר ששמו name. אם לא קיים איש קשר בשם זה, מוחזר הערך null |
| String[] getAllContactsNames() | הפעולה מחזירה את אוסף השמות של אנשי הקשר שקיימים בספר הטלפונים. אם ספר הטלפונים ריק, מוחזר מערך בגודל אפס |
| String toString() | הפעולה מחזירה מחרוזת המתארת את ספר הטלפונים במבנה הזה: name1 phone1 name2 phone2 : פרטי אנשי הקשר יופיעו בשורות נפרדות ממוינות בסדר אלפביתי לפי שמם |

לפניכם דוגמה לספר טלפונים. הספר נבנה, לתוכו מוכנסים 6 אנשי קשר ובסוף הוא מודפס:

```
// יצירת ספר טלפונים
PhoneBook pb = new PhoneBook();

// הוספת אנשי קשר לספר הטלפונים
pb.addContact("Galit Israel", "03-9089730");
pb.addContact("Avner Cohen", "02-7474747");
pb.addContact("Gershon Avraham", "02-8900011");
pb.addContact("Daniela Yariv", "04-5677708");
pb.addContact("Alice Marlo", "04-5699300");
pb.addContact("Bob Denver", "04-5699300");
pb.addContact("Galit Israel", "02-6412222");

// הדפסת ספר הטלפונים
System.out.println(pb);
```

הפלט המתקבל:

| | |
|------------------------|-------------------|
| <i>Alice Marlo</i> | <i>04-5699300</i> |
| <i>Avner Cohen</i> | <i>02-7474747</i> |
| <i>Bob Denver</i> | <i>04-5699300</i> |
| <i>Daniela Yariv</i> | <i>04-5677708</i> |
| <i>Galit Israel</i> | <i>02-6412222</i> |
| <i>Gershon Avraham</i> | <i>02-8900011</i> |

מה עליכם לעשות?

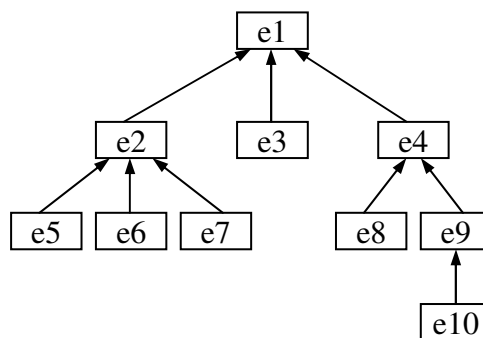
- א. כדי לכתוב יישום ממוחשב לספר הטלפונים עליכם להגדיר לאילו מחלקות תזדקקו, מהן תכונותיהן ומה ממשקן של המחלקות. הגדירו את המידול שאליו הגעתם בעזרת איורי UML.
- ב. ממשו את המחלקה PhoneBook.
- ג. לשם בדיקת המחלקה, הריצו את התוכנית הראשית TestPhoneBook.java הנמצאת בתיקייה HelpFiles.
- ד. נתחו את יעילות פעולות המחלקה PhoneBook שכתבתם בסעיף ב.

מהצלחה!

פרק 6 דף עבודה מס' 6 עצם שתכונתו מטיפוס עצמו

א. השאלה הבאה מתייחסת למחלקה `Bead` שהוצגה בסעיף ח:
כתבו פעולה חיזונית המקבלת חרוז ומדפיסה את כל השרשרת הקשורה אליו (כדי למנוע לולאה אינסופית במקרה של מעגל, הגבילו את מספר ההדפסות ל-15).

ב. השאלות הבאות מתייחסות למחלקה `Employee` שהוצגה בסעיף ח.1:
1. כתבו קטע תוכנית שייצור עובדים (ששמותיהם `e1, e2, ... e10`) לפי המבנה הארגוני שלפניכם:



2. ממשו את הפעולה:

```
public static boolean isBoss(Employee emp)
```

הפעולה מקבלת עובד בחברה ומחזירה "אמת" אם הוא מנהל החברה (הבוס), ו"שקר" אחרת.

3. ממשו את הפעולה:

```
public static void printBossName(Employee emp)
```

הפעולה מקבלת עובד בחברה (שאינו מנהל), ומדפיסה את שם מנהל אותה החברה (הבוס).

4. ממשו את הפעולה:

```
public static int getEmployeeLevel(Employee emp)
```

הפעולה מקבלת עובד בחברה ומחזירה מספר המתאר את הרמה שלו במבנה הארגוני. מנהל החברה נמצא ברמה 1 במבנה הארגוני, העובדים הכפופים אליו באופן ישיר נמצאים ברמה 2, וכך הלאה. במבנה הארגוני המתואר באיור בשאלה הראשונה, העובדים `e5` ו-`e8` נמצאים באותה רמה – רמה 3, ואילו העובד `e10` נמצא ברמה 4.

5. א. הסבירו למה לא ניתן לממש את הפעולות הבאות:

1. `public static void printSubEmployees(Employee emp)`

הפעולה מקבלת עובד ומדפיסה את שמות כל העובדים הכפופים לו באופן ישיר.

2. `public static boolean isSupervisor(Employee emp)`

הפעולה מקבלת עובד ומחזירה "אמת" אם הוא אחראי על עובד אחד לפחות.

ב. כיצד ניתן לשנות את הייצוג של `Employee`, כך שניתן יהיה לממש את הפעולות המופיעות בסעיף א.

מהצחקה!