פרק 6
הפניות ועצמים מורכבים

1. הזמן הנדרש

10 שעות.

2. מטרות

בפרק זה נדונים כמה תכנים הקשורים ביניהם, ועם זאת, כל אחד מהם הוא בעל מורכבות הדורשת תשומת לב מיוחדת בהוראה. נפרט את התכנים:

1. מהותו של משתנה מטיפוס מחלקה והאופן שהוא מכיל עצם מטיפוס המחלקה. משמעות של כמה הפניות לעצם.
2. "עצם מורכב" וחידוד הפעולות הקשורות בתכונה מטיפוס מחלקה.
3. ייצוג אוסף וניהולו באמצעות מערך.
4. העתקות: העתקה שטוחה (shallow copy) לעומת העתקה עמוקה (deep copy).
5. עצם מורכב שתכונתו מטיפוס עצמו, כמבוא מוחשי לשרשרת חוליות.

3. מבנה הפרק

א – אתחול של משתנה מטיפוס מחלקה

יש להבהיר לתלמידים במלל ובעזרת איורים מרובים שכשמדברים על עצם מדברים למעשה על שלושה רכיבים: **משתנה** מטיפוס המחלקה, המכיל **הפניה** לעצם וה**עצם** עצמו. לפחות בתחילת לימוד הפרק יש להקפיד על שיח מדויק שיבחין בין הרכיבים ויגדיר בדיוק כל אחד מהם. התחושה, שקיומם של שלושת הרכיבים הללו הוא נכון, תלך ותתחזק לאורך הפרק בעזרת הדוגמאות השונות שיוצגו.

הסעיף דן בשלושה סוגי אתחול של משתנה מטיפוס מחלקה:

סוג 1 – אתחול על ידי הפניה לעצם חדש. ההפניה נוצרה בתום הפעולה **new**. באתחול כזה קיימת הפניה אחת אל העצם החדש. את האתחול הזה התלמידים כבר פגשו ביסודות או לפחות בפרק 2 - עצמים.

סוג 2 – אתחול על ידי הפניה קיימת. בדרך זו נוצרות כמה הפניות לאותו העצם. אחת ממטרות הפרק היא להבהיר ולהדגיש את רמת הזהירות והדיוק הנדרשת בעבודה עם הפניות אלה ובאמצעותן.

סוג 3 – הצבת הערך **null**, שהוא ערך הפניה תקני.

שם משתנה מטיפוס מחלקה המכיל הפניה לעצם, אינו תכונה של העצם. יכול להיות בהחלט כי קיימים שני משתנים שונים המכילים הפניה לאותו עצם, ואז לכאורה יש לעצם שני שמות שונים. לכן חשוב להדגיש כי השמות אינם שייכים לעצם, אלא הם שמות חיצוניים. למרות זאת, לנוחיות

הדיון, נשתמש בביטויים מהסוג: "העצם b1" במקום הביטוי המסורבל והמדויק יותר: "העצם שמשתנה b1 מפנה אליו".

ב – משתנה מטיפוס מחלקה שאינו מאותחל

ניתן להכריז על משתנה מטיפוס מחלקה בלא אתחול. כאשר משתנה מטיפוס מחלקה אינו מאותחל, אין בו ערך כלל גם לא הערך **null**.

יש להבחין בין משתנה המוגדר כתכונה של מחלקה, שאז כברירת מחדל הוא מאותחל ל-**null**. לבין משתנה מטיפוס מחלקה המוגדר כמשתנה מקומי בפעולה כלשהי, ואם לא אותחל מפורשות אזי הוא נשאר בלתי מאותחל.

ההבדל בין משתנה מטיפוס מחלקה שערכו **null**, לבין משתנה שכזה שלא אותחל, מתבטא בסוג השגיאה המתקבל בעת ניסיון לפנות לעצם (שהמשתנה מפנה אליו לכאורה), כדי שיפעיל את פעולותיו. אם המשתנה מכיל **null** אזי מתקבלת השגיאה: NullPointerException והיא שגיאת זמן ריצה. לעומת זאת אם המשתנה לא מאותחל, תתקבל שגיאת הידור המודיעה כי המשתנה לא אותחל.

ג – השוואת הפניות

השוואה בין משתנים מטיפוס מחלקה היא למעשה ההשוואה של הפניות.

<u>תשובה לשאלת המחשבה המופיעה בסעיף זה:</u>

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

תשובה:

בג'אווה:

```
if (b1.getCapacity() = b2.getCapacity() &&
                b1.getCurrentAmount() == b2.getCurrentAmount())
      return true;
else
      return false;
```

בסישרפ:

```
if (b1.GetCapacity() = b2.GetCapacity() &&
                b1.GetCurrentAmount() == b2.GetCurrentAmount())
      return true;
else
      return false;
```

ד – מערך של עצמים

חשוב להבין כי 'מערך של עצמים' הוא עצם מטיפוס מערך המכיל בתאיו הפניות לעצמים.
למשל:

```
Box arr[] = new Box[10];
// אתחול תא במערך
Arr[0] = new Box(1, 2, 3);
Box b1 = arr[0];
```

כעת אם נשנה את התכונות של העצם ש-b1 מפנה אליו:

```
b1.setLength(4);
```

ובסישרפ:

```
b1.SetLength(4);
```

נוכל לראות את השינוי על ידי אחזור האורך דרך arr[0]. כלומר ההפניות b1 ו-arr[0] מפנות שתיהן לאותו עצם.

ה – עצם מורכב

- נושא העצם המורכב כולל אספקטים נוספים פרט לנושא ההפניות. הרעיון שקיימת תכונה שהיא עצמה עצם עלול להיות קשה לתלמידים. יש לוודא שהם אכן מבינים כיצד לנהל את העצם המורכב: הפעלת פעולות של התכונה שהיא מטיפוס מחלקה, היחס בין שני העצמים (העצם המורכב והתכונה) וכדומה.
- מחלקות שתכונותיהם מטיפוס מחרוזת מגדירות אף הן עצמים מורכבים, אך כיוון שמחרוזת היא עצם מיוחד והיא שונה משאר העצמים, לא מומלץ ללמד את נושא העצם המורכב בעזרת הטיפוס המחרוזתי.
- המחלקות 'הורה' ו'תלמיד' הן אבני הבסיס לדוגמה שמכאן והלאה תלווה את לימוד הפרקים הדנים באוספים. לכן חשוב מאוד שהתלמידים יממשו את המחלקות המוזכרות בפרק (הורה, תלמיד). כך הם יוכלו לענות על כל השאלות התיאורטיות בפרק (ויש כאלה הרבה יחסית לפרקים אחרים), ולבדוק את תשובותיהם באמצעות קוד. כמו כן הם יוכלו מכאן ואילך לשנות את המימוש של הפעולות ואת ייצוג המחלקות עצמן ללא שינוי הממשק, על פי תכני הפרקים. דבר זה ידגים וימחיש להם את עקרונות ההכמסה והסתרת המידע.
- סעיף זה חשוב מאוד כהכנה לקראת העיסוק ברשימה הכיתתית, שגם היא עצם מורכב (התכונה שלה היא מטיפוס המחלקה Student).
- בסעיף זה מושם דגש על פעולות שבהן הפרמטר (שהוא עצם), הנשלח לפעולה הבונה, מוצב בתכונה של המחלקה או שהתכונה של העצם המורכב, שהיא מטיפוס מחלקה, מוחזרת כערך החזרה. בשני המקרים אם לא מבצעים העתקה, נוצר מצב של שתי הפניות לעצם אחד. בסעיף זה חשובה ההבנה של מהלך הזימון של הפעולה ומצבם של המשתנים המקומיים של הפעולה, בכל שלב.
- שאלה למחשבה: אם כותרת הפעולה הבונה של Student הייתה מוגדרת כך ששני הפרמטרים הראשונים הם שם התלמיד ומספר הטלפון בביתו ושני האחרונים הם שם ההורה ומספר הטלפון הנייד שלו, באופן הזה:

בג׳אווה:

```
public Student(String stuName, String phoneNum,
                                    String parName, String cellNum)
```

בסישרפ:

```
public Student(string stuName, string phoneNum,
                                    string parName, string cellNum)
```

היכן היה נוצר העצם מטיפוס Parent? נסו לממש את הפעולה הבונה במקרה זה.

שאלה זו מראה כי ניתן ליצור עצם, שהוא תכונה של העצם המורכב, בתוך הפעולה הבונה של העצם המורכב ולא לקבלו כפרמטר. במקרה זה אנו צריכים לקבל כפרמטר את כל פרטי העצם, אך לא תישאר בידינו רשומת הורה נפרדת במאגר ההורים!

- בסעיף זה אנו מדגישים את העובדה שעצם מועבר על ידי ההפניה אליו. אצל המורים עלולים לבלבל בין המושגים ״by value״, ״by reference״, הזכורים משפות אחרות. בפועל, בג׳אווה גם עצמים וגם טיפוסים בסיסיים מועברים על ידי העתקת ערך, כלומר ״by value״. (אם כי ה-value במקרה של עצם הוא של ה-reference...). במקרה של משתנה מטיפוס מחלקה הערך הוא הפניה, אך גם הוא מועתק בדיוק כמו טיפוס בסיסי. את נושא ההעברה by reference לא הזכרנו בפרק, אם כי הוא קיים בשפת סישרפ (בג׳אווה הוא כלל אינו קיים). המלצה: עדיף לא להזכיר כלל מושגים אלו בכיתות.
- בתרשימי עצמים, כאשר ערך תכונה מסוימת קיים אך אינו חשוב לעצם הדיון, הוא צבוע באפור וכך נחסך פירוט מיותר.

<u>תשובות לשאלות המחשבה המופיעות בסעיף זה:</u>

? 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).

תשובה:

נשנה את מספר הטלפון הנייד דרך p1, ונבדוק האם גם הטלפון הנייד של התכונה parent של st1 השתנה. אם כן, אזי שתי ההפניות פונות לאותו העצם. אם לא, הוכחנו שההפניות פונות לשני עצמים שונים:

בג׳אווה:

```
p1.setCellNumber(”056-789546”);
System.out.println(st1.getParent().getCellNumber());
```

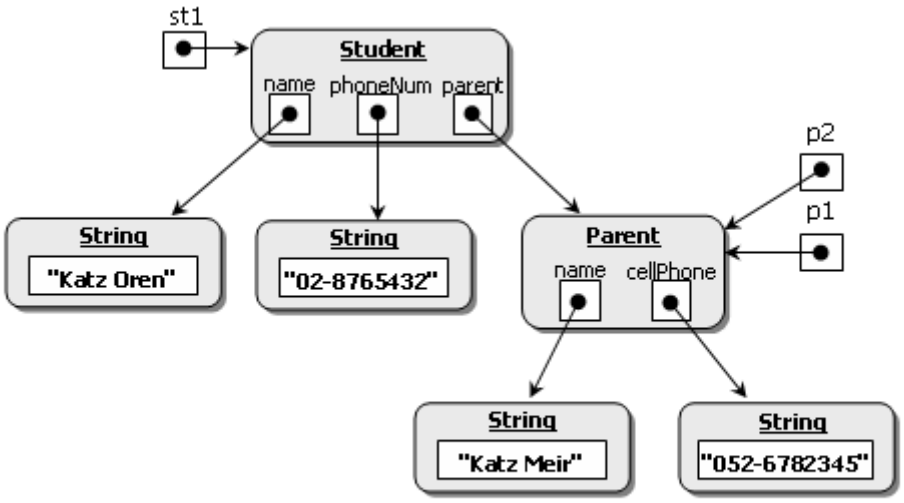
בסישרפ:

```
p1.SetCellNumber(”056-789546”);
Console.WriteLine(st1.GetParent().GetCellNumber());
```

בפועל, כאשר נריץ את הקוד הזה, נראה שיש שינוי בערך התכונה המשתקף בהדפסה דרך המשתנה st1, כלומר הוכחנו ששתי ההפניות פונות לאותו העצם.

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

תשובה:



ו – עצם מורכב המייצג אוסף

- נושא זה הוא לכאורה המשך ישיר של הסעיף הקודם, שהרי גם עצם שתכונתו מערך הוא עצם מורכב, ואולם בסעיף זה נידונים תכנים נוספים. בסעיף זה אנו מראים ניהול אוסף באמצעות מערך, כמבוא לניהול אוסף באמצעות שרשרת חוליות, שיוצג בפרק הבא. התלמיד מתנסה בהכנסה והוצאה לאוסף המיוצג וממנו על ידי מערך. בתוך כך התלמיד נתקל בצורך לצמצם רווחים בהוצאה ולהזיז איברים ימינה בהכנסה. הבנת שלב זה משמשת מבוא טוב להבנת היתרון שבייצוג אוסף באמצעות חוליות.
- מטרת הסעיף היא להציג מימוש של אוסף באמצעות מערך. הסעיף מאפשר לדון בחסרונות של ייצוג אוסף על ידי מערך.
- עשויה לעלות השאלה מדוע כדאי להגדיר את האוסף StudentList, ולא להשתמש במערך ממוין ישירות בתוך הפעולה הראשית (הפעולה שמבצעת המזכירה ליצירת רשימה כיתתית והדפסתה). ברור שאם הרשימה תהיה רק מערך בפעולה שהמזכירה משתמשת בה כדי ליצור ולהדפיס רשימה כיתתית, הרי שמיד אחרי ההדפסה וסיום הפעולה הרשימה אינה קיימת. אם רוצים רשימה שקיימת לאורך זמן, כך שניתן יהיה להשתמש בה באופנים שונים, צריך לייצר עצם מתאים, ומכאן שצריך מחלקה.
- נרחיב את השאלה והתשובה שהוצגו בסעיף הקודם: יכולה לעלות השאלה, מדוע בכלל להגדיר את האוסף StudentList, ולא להשתמש במערך ממוין ישירות בתוך הפעולה שמבצעת המזכירה ליצירת רשימה כיתתית והדפסתה. בגישה זו, המערך ימוין בפעולה עצמה. התשובה היא שהרשימה הכיתתית היא אוסף שצריך להישמר לאורך זמן, כך שניתן יהיה להשתמש בו למטרות שונות לאורך שנת הלימודים. במערכת אמיתית, סביר שהרשימה תשמר במסד נתונים. בגישה הפשטנית יותר שלנו, היא נשמרת כעצם, מופע של המחלקה. גם המיון, עדיף שיתבצע על ידי העצם, ולא על ידי פעולה המשתמשת בו. עקרון חשוב של תכנות

מונחה עצמים הוא להגדיר מחלקות עבור נתונים חשובים של היישום, ולרכז במחלקה כזו את הקוד לביצוע פעולות שימושיות על העצמים מטיפוס המחלקה. ריכוז הקוד פירושו שהקוד נכתב פעם יחידה במקום יחיד, במקום פעמים רבות במקומות שונים, וזה כמובן מאפשר שימוש חוזר בקוד שנכתב ונבדק היטב.

- פעולות ההכנסה וההוצאה מרשימה כיתתית נקראות: "הוספה" – add(…); ו"מחיקה" – del(…), כדי למנוע בהמשך, בפרק רשימה, כפילות ובלבול בין פעולות אלה לבין הפעולות insert(…) ו-remove(…) של רשימה, שבאמצעותן נממש את פעולות הרשימה הכיתתית.
- יש לחזור ולהדגיש כי על פי עולם הבעיה שעיצבנו, אי אפשר להכניס מעבר לשלושים וחמישה תלמידים לכיתה, האחראית על בדיקה זו היא המזכירה, שאחד מתפקידיה הוא לשמור את מספר התלמידים בכיתה. אך כאשר הפעולה מזומנת, נניח תמיד שיש מקום להוסיף תלמיד לרשימה. למעשה, זו הנחה סמויה העומדת מאחורי הפעולה add(…). כלומר מי שמזמן את הפעולה אחראי על בדיקה זו, ולא הפעולה עצמה. ביישום אמיתי, הפעולה הייתה בודקת בכל הכנסת תלמיד אם מספר התלמידים עדיין קטן מ-35, כלומר האם המערך אינו מלא. לשם הפשטות, העברנו אחריות זו למזכירה, ולא כללנו בדיקה זו בקוד.
- יכולנו לבחור ממשק אחר לפעולת ההוספה שבו לא היינו עוסקים בעצמים, למשל כך:

ג'אווה:
```
void add(String name, int phone)
```
סישרפ:
```
void Add(string name, int phone)
```

אך אנו מעדיפים את הממשק המקבל עצמים כדי שנוכל לדון בנושאים המעניינים אותנו. יש מקום לדון בכיתה בשני הממשקים, ובשאלה היכן נוצר העצם Student בשני המקרים.

- טיפוס הנתונים: רשימה כיתתית – הוא טיפוס נתונים מופשט. נושא זה של טיפוס נתונים מופשט אינו פשוט ואנו נעסוק בו בהרחבה בפרקים הבאים. בפרק זה מומלץ להזכיר את העובדות הבאות אך לא את המושג טיפוס נתונים מופשט. מושג זה מוזכר לצורך המורים בלבד:

כיוון שאין במחלקה Student פעולה המשנה את שם התלמיד, לא ניתן לפגוע במבנה הפנימי של הרשימה הכיתתית (שבה אין שני תלמידים בעלי אותו שם. אם הרשימה מוחזקת באופן ממוין אזי גם אין דרך לפגוע במיון, ועל כך נדון בהמשך, בסעיפים ו.3., ו.5.).

אם בממשק המחלקה היו מוגדרות פעולות המשנות את תכונות המחלקה Student, אזי האוסף לא היה מופשט. מדוע?

נניח כי המתכנת מחליט לשמור על מיון המערך לאורך כל התוכנית, ולכן הוא מכניס את התלמידים למקום המתאים במערך תוך שמירה על הסדר האלפביתי.

אך מה יקרה לאחר ביצוע הקוד הזה:

בג'אווה:

```
public static void main(String[] args)
{
      .....//יצירת המופעים של ההורים
      StudentList lst = new StudentList ();
      Student st1 = new Student(”Potter Harry”, p1, 02996187);
      Student st2 = new Student(”Scofield Michael”, p2, 08945632);
      lst.add(st1);
      lst.add(st2);

      st1.setName(”Weizmann”);
      System.out.println(lst);
}
```

בסישרפ:

```
public static void Main(string[] args)
{
      .....//יצירת המופעים של ההורים
      StudentList lst = new StudentList ();
      Student st1 = new Student(”Potter Harry”,p1, 02996187);
      Student st2 = new Student(”Scofield Michael”, p2, 08945632);
      lst.Add(st1);
      lst.Add(st2);

      st1.SetName(”Weizmann”);
      Console.WriteLine(lst);
}
```

הרשימה המודפסת לא תהיה ממוינת:

Weizmann 02996187

Scofield 08945632

כלומר מבנה הרשימה נהרס שלא דרך פעולות הרשימה עצמה. כאשר ניתן להרוס את המבנה של טיפוס מסוים, שלא דרך הפעולות הטיפוס, אנו אומרים כי טיפוס זה אינו מופשט. לכן מנענו את האפשרות לשנות את שם התלמיד, כדי שהאוסף יישאר טיפוס נתונים מופשט. חיזוק נוסף לטענה שרשימה כיתתית היא טיפוס נתונים מופשט הוא העובדה שהצגנו שני מימושים שונים למחלקה. שני המימושים מציעים אותן פעולות, ושומרים על התכונות של המחלקה, לכן משתמש במחלקה אינו יכול להבדיל ביניהם. אם נחליף מימוש אחד במשנהו, הקוד שכתב המשתמש ימשיך לרוץ. האפשרות להחלפת מימוש תוך שמירת הממשק ותכונות המחלקה היא אפיון חשוב של טיפוס נתונים מופשט.

- היעילות של הפעולה getStudent(...) / GetStudent(...) משתנה בהתאם להחלטה האם שומרים את המערך ממוין באופן תמידי או מממיינים אותו רק כאשר מממשים את הפעולה toString()/ToString().

אם המערך ממוין, אזי יעילות הפעולה getStudent(...) / GetStudent(...) היא O(logn), שכן החיפוש הוא בינרי, אחרת היעילות תהיה O(n).

<u>תשובות לשאלות המחשבה המופיעות בסעיף זה:</u>

? כתבו פעולה המקבלת רשימה כיתתית ושם של תלמיד, ומדפיסה את מספר הטלפון הנייד של ההורה האחראי על תלמיד זה.

תשובה:

בג'אווה:

```
public static void printCelNum(StudentList lst, String name)
{
      Student st1 = lst.getStudent(name);
      System.out.println(st1.getParent().getCellNum());
}
```

בסישרפ:

```
public static void PrintCelNum(StudentList lst, string name)
{
      Student st1 = lst.GetStudent(name);
      Console.WriteLine(st1.GetParent().GetCellNum());
}
```

? יותם, הבכור באחי משפחת כהן, התייצב במזכירות בית הספר ודיווח על שינוי מספר הטלפון הנייד של אביו. כיצד תבצע נטע המזכירה את העדכון? פרטו ונמקו בעזרת קוד או בעזרת תרשימי עצמים.

תשובה:

נניח כי המספר החדש הוא: "052-5436789"

נטע תכתוב את הקוד הזה, בג'אווה:

```
Student yotam = lst.getStudent("Yotam Cohen");
yotam.getParent().setCellNum("052-5436789");
```

ובשפת סישרפ:

```
Student yotam = lst.GetStudent("Yotam Cohen");
yotam.GetParent().SetCellNum("052-5436789");
```

יש לשים לב, כי המספר יתעדכן אצל כל האחים של יותם, כיוון שכולם מפנים לאותו עצם של הורה שיותם שינה את מספר הטלפון הנייד שלו.

? כתבו את פעולת ההוספה ללא הגדרת התכונה lastPosition. נתחו את זמן הריצה של פעולת ההוספה בשני מקרים: עם התכונה ובלעדיה. נסחו את מסקנותיכם.

תשובה:

בג'אווה:

```
public void add(Student st)
{
      int i = 0;
      while (i< this.list.length)
      {
            if (this.list[i] != null)
                  i++;

      }
      this.list[i] = st;
}
```

בסישרפ:

```
public void Add(Student st)
{
      int i = 0;
      while (i< this.list.Length)
      {
            if (this.list[i] != null)
                  i++;

      }
      this.list[i] = st;
}
```

ניתוח יעילות מצביע על כך שכדאי להוסיף את התכונה למחלקה.

יעילות הפעולה: עם התכונה lastPosition, היעילות היא O(1).

ללא התכונה lastPosition, היעילות היא O(n).

? נניח כי הרשימה הכיתתית lst קיימת. במהלך תוכנית כלשהי, הוצבה במשתנה student1 הפניה לתלמיד ואז ביצעה התוכנית את הזימון הזה:

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

א. התבוננו בפעולה זו:

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

כמה הפניות לתלמיד החדש קיימות לפני ביצוע שורת הקוד הבאה בגוף פעולת ההוספה, וכמה קיימות אחריה. הסבירו את תשובתכם בפירוט.

ב. כמה הפניות לתלמיד יישארו עם תום פעולת ההוספה?

תשובה:

א. לפני ביצוע הפעולה קיימות ההפניות האלה: student1 ו-st

(st הוא הפרמטר של הפעולה add(…) / Add(…) , ולכן הוא ההפניה המקומית של הפעולה. ההפניה נוצרת בעת זימון הפעולה).

אחרי ביצוע שורת הקוד הנתונה, מתוספת לשתי ההפניות הקודמות עוד הפניה:

`this.list[this.lastPosition]`

במהלך הפעולה קיימות שלוש הפניות סך הכול.

ב. עם תום הפעולה ההפניה st נעלמת, שכן היא הייתה הפניה מקומית של הפעולה. לפיכך, נשארות רק שתי הפניות: student1 ו-`this.list [this.lastPosition]`.

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

תשובה:

האלגוריתם מתאים גם למקרי קצה. שאלה זו באה כהכנה לפעולות על שרשרת חוליות, שם יש התנהגות מיוחדת במקרי קצה, וכדאי שהתלמידים יתרגלו לבדוק מקרי קצה בפעולות.

? בג'אווה: ממשו את הפעולה **void** moveOthers (**int** place).

בסישרפ: ממשו את הפעולה **void** MoveOthers (**int** place).

תשובה:

שימו לב כי הפעולה יכולה להיות פרטית, כיוון שהיא משמשת רק כפעולת עזר של האלגוריתם. נזכיר שאין צורך לבדוק האם יש מקום במערך, כיוון שההנחה היא שהפעולה לא תזומן אם אין בו מקום:

בג'אווה:

```
private void moveOthers(int place)
{
      for (int i = this.lastPosition; i > place; i--)
      {
            this.list[i] = this.list[i-1];
      }
}
```

בסישרפ:

```
private void MoveOthers(int place)
{
      for (int i = this.lastPosition; i > place; i--)
      {
            this.list[i] = this.list[i-1];
      }
}
```

? כיוון שהמערך ממוין, עולה השאלה האם ניתן להיעזר בחיפוש בינרי ולהקטין את סדר הגודל של פעולת ההוצאה ל-O(logn)? הסבירו ונמקו את תשובתכם.

תשובה:

לא ניתן להיעזר בחיפוש בינרי ולהקטין את סדר הגודל של פעולת ההוצאה ל-O(logn). אף שפעולת מציאת האיבר המבוקש להוצאה אכן מהירה יותר, יעילותו של החיפוש הבינרי היא O(logn), משום שאחרי ביצוע פעולת ההוצאה יש צורך "לצמצם את הרווח" שנפער במערך, כלומר להזיז את כל האיברים שמאלה. לפיכך, זמן ריצה של הפעולה יישאר מסדר גודל O(n).

ז – העתקה

בסעיף זה נעסוק בהעתקה שטוחה לעומת העתקה עמוקה. לא קראנו למושגים האלו בשמם משום שאין צורך בהעמסת מונחים, די שהתלמידים יבינו את ההבחנה בין השניים. בכיתות החזקות אפשר להציג לתלמידים את המושגים המגדירים את שתי הצורות של ההעתקה. התעסקות במושגים אלה, בנוסף להרחבת הידע בנושא סוגי ההעתקות, משמשת כתרגול נוסף להבחנה בין הפניות לעצמים.

העתקה של עצם מורכב – באופן שבו כל תכונותיו שהן עצמים אכן מועתקות לעצמים חדשים ולא מתבצעת העתקת הפניות בלבד (וכן עבור תכונות של עצמים אלו, בכל עומק) – נקראת **העתקה עמוקה** (**deep copy**). העתקה של עצם שמבצעת העתקה של תכונות העצם רק על ידי העתקת ההפניות היא **העתקה שטוחה** (**shallow copy**).

אם נרצה שמחלקה תכיל שתי פעולות בונות מעתיקות, שאחת מהן מבצעת העתקה עמוקה והאחרת העתקה שטוחה, נצטרך לשנות את רשימת הפרמטרים בכותרות הפעולות, כיוון שלשתי הפעולות יש אותה הכותרת, למשל על ידי תוספת פרמטר דגל, שיציין האם ההעתקה תהיה שטוחה (העתקה של ההפניה עצמה) או העתקה עמוקה.

ברור שאין הבדל בין העתקה שטחית לעמוקה אם כל התכונות במחלקה הן מטיפוסים בסיסיים. כאשר מחלקה מגדירה עצמים מורכבים, כלומר יש בה תכונות מטיפוסי מחלקה יש הבדל בין שתי ההעתקות. העתקה עמוקה קשה לתכנות אם יתכן שבאוסף העצמים יש סדרת הפניות שמהווה מעגל. למשל, אם נוסיף לכל תלמיד הפניה לתלמיד אחר (נניח חבר טוב שלו), אזי יתכן שעצם של תלמיד א' מחזיק הפניה לעצם של תלמיד ב', וזה מצדו מחזיק הפניה לעצם של תלמיד א'. תכנות פשטני יגרום כאן לתכנית שאינה מסתיימת, כיון שבהינתן עצם של תלמיד א', התכנית תקרא לעצמה להעתקת תלמיד ב', זימון זה יזמן שוב את הפעולה להעתקת תלמיד א', וחוזר חלילה. תכנית נכונה צריכה לזהות מבנים עם מעגלים ולהעתיק כל עצם פעם אחת. דבר זה הוא מעבר ליכולות של תלמידי יחידה זו. בדוגמה שלנו בעיה זו אינה קיימת.

כמובן, גם כאשר אין מעגלים, ביצוע העתקה עמוקה דורש שלמחלקות של העצמים שהם תכונות תהיה פעולה בונה מעתיקה וזו דרישה שאינה קיימת ביחידת הלימוד.

אין צורך לדון בנושא זה באריכות בכיתה, אך יש להיזהר מקביעה שאחרי לימוד סעיף זה אנו מבינים לגמרי איך כותבים העתקה עמוקה.

תשובה לשאלת המחשבה המופיעה בסעיף זה:

? כמובן 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?

תשובה:

כאשר נבצע את ההעתקה של הרשימה, נעתיק את התלמידים העתקה עמוקה: כלומר גם את העצמים מטיפוס Parent, נעתיק ממש (ולא נעתיק רק את ההפניות להורים). בצורה זו ייווצר נתק מוחלט בין שתי הרשימות, ולא ניתן יהיה לעדכן את פרטי ההורים ברשימה הבית ספרית מתוך הרשימה של מדריך הטיולים.

ח – עצם שתכונתו מטיפוס עצמו

סעיף זה משמש כמבוא לפרק הבא העוסק במבני חוליות. חוליה היא עצם מורכב, אשר תכונה אחת שלה (או יותר), היא מטיפוס העצם עצמו. מבני חוליות ישמשו אותנו לשמירה וטיפול באוספים, ולכן כדאי שהתלמידים יכירו אותם היטב, אך לא רק בהקשר לחוליות. בסעיף זה התלמידים יבינו את הרעיון ברמה מוחשית (חרוזים ושרשרת חרוזים), עוד לפני הדיון בחוליה ובשרשרת חוליות.

יש להתעכב על הרעיון של משתנה מטיפוס מחלקה המשמש לסריקה. רעיון זה חדש לתלמידים והוא לא תמיד פשוט להבנה. עד כה התלמידים ראו משתנה מטיפוס מחלקה שהוגדר לצורך החזקת עצם קבוע לאורך התוכנית (לעתים שינינו את ההפניה שבתוך משתנה מטיפוס מחלקה, אך לא כמטרה בפני עצמה). לעומת זאת, משתנה מטיפוס מחלקה המשמש לסריקה אינו מפנה לעצם קבוע מסוים אלא מפנה לעצמים מתחלפים לצורך סריקה.

תשובות לשאלות המחשבה המופיעות בסעיף זה:

? כתבו קטעי קוד עבור שרשרות החרוזים המתוארות להלן:

א. הרכיבו שתי שרשרות שונות, הזהות 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.

תשובה:

את המחרוזות ניתן ליצור בשתי דרכים. ובין שתיהן אין שום שוני מבחינת המבנה המתקבל:

דרך א: מהסוף להתחלה.

דרך ב: מהתחלה לסוף.

כדי להדגים את שתי הדרכים ניצור את השרשרת הראשונה מהסוף להתחלה ואת השנייה מההתחלה לסוף. כמובן שניתן ליצור את שתי המחרוזות באותו האופן.

דרך א: יצירת השרשרת הראשונה מהסוף להתחלה:

```
Bead b1 = new Bead("Green", null);
b1 = new Bead("Yellow", b1);
```

```
b1 = new Bead("Red", b1);
b1 = new Bead("Green", b1);
b1 = new Bead("Yellow", b1);
```

שימו לב את ההפניה b1 ניתן לעדכן שוב ושוב, כיוון שקודם מתבצע האגף השמאלי של ההשמה ורק לאחר מכן מתבצעת ההשמה עצמה. מובן שאם התלמידים השתמשו לכל חרוז בהפניה אחרת, יש לקבל זאת.

דרך ב: יצירת השרשרת השנייה מההתחלה לסוף:

בג'אווה:

```
Bead b2 = new Bead("Yellow", null);
b2.setNextBead(new Bead("Green", null));
b2.getNextBead().setNextBead(new Bead("Red", null));
b2.getNextBead().getNextBead().setNextBead
                                        (new Bead("Yellow", null));
b2.getNextBead().getNextBead().getNextBead().setNextBead
                                          (new Bead("Green",null));
```

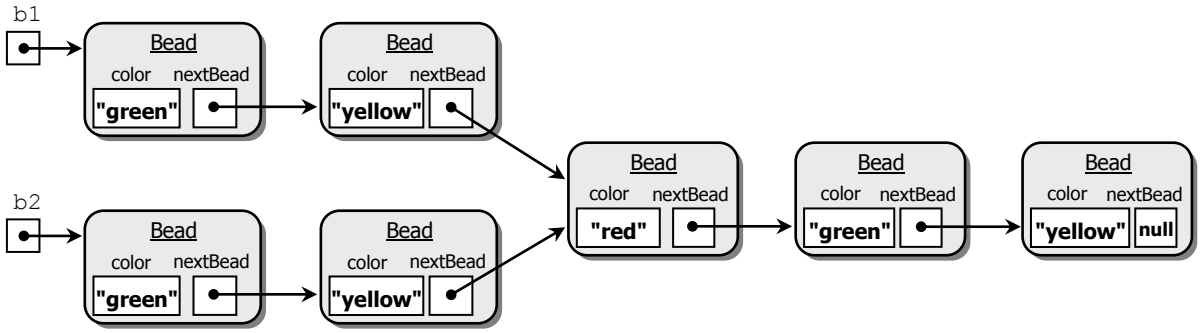
בסישרפ:

```
Bead b2 = new Bead("Yellow", null);
b2.SetNextBead(new Bead("Green", null));
b2.GetNextBead().SetNextBead(new Bead("Red", null));
b2.GetNextBead().GetNextBead().SetNextBead
                                        (new Bead("Yellow", null));
b2.GetNextBead().GetNextBead().GetNextBead().SetNextBead
                                          (new Bead("Green",null));
```

ניתן להתרשם כי הצורה הראשונה, יצירת שרשרת מהסוף להתחלה, נוחה יותר.

המשך השאלה:

חברו את שתי השרשרות שיצרתם על ידי שינוי הפניה בשרשרת השנייה, כך שיתקבל מבנה כמתואר באיור שלפניכם (שימו לב שאחרי ביצוע המשימה, אין בתוכניתכם אף הפניה לשלושת החרוזים האחרונים שבשרשרת השנייה).



תשובה:

ניתן לעשות זאת בפקודה אחת (בהנחה ש-b1 ו-b2 מפנים לתחילת השרשרות):

בג'אווה:

```
b2.getNextBead().setNextBead(b1.getNextBead().getNextBead())
```

כמובן שאפשר לפצל את הפקודה הזו לכמה פקודות. נראה זאת בשפת סישרפ:

```
Bead pos1 = b1;
pos1 = pos1.GetNextBead();
Bead pos2 = b2;
pos2 = pos2.GetNextBead();
pos2.setNextBead (pos1.getNextBead())
```

המשך השאלה:

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

תשובה:

זוהי הזדמנות להראות שניתן ליצור הפניה שתסרוק את כל השרשרת עד סופה, ושבה נוכל להיעזר כדי לחבר את שתי השרשרות. גם בסעיף זה לא חייבים ליצור הפניה לסריקה, אך זו הזדמנות להכיר את הדרך הזו לסריקה ולשימוש בהפניות:

בג'אווה (בהנחה שנתונים b1 ו-b2 המפנים לשרשרות המקוריות):

```
Bead p1 = b1;
while (p1.getNextBead()!= null)
       p1 = p1.getNextBead();

p1.setNextBead(b2);

while (p1.getNextBead()!= null)
       p1 = p1.getNextBead();

p1.setNextBead(b1);
```

בסישרפ:

```
Bead p1 = b1;
while (p1.GetNextBead()!= null)
       p1 = p1.GetNextBead();

p1.SetNextBead(b2);

while (p1.GetNextBead()!= null)
       p1 = p1.GetNextBead();

p1.SetNextBead(b1);
```

ט – מחרוזת עצם מקובע

- כיוון שמחרוזת היא עצם שאינו ניתן לשינוי (**immutable**), כל פעולות הממשק העוסקות בשינוי מחרוזת מייצרות מחרוזת חדשה. למשל, הפעולה שהופכת את כל האותיות במחרוזת לאותיות גדולות, אינה משנה את המחרוזת שמזמנת את הפעולה, אלא מחזירה מחרוזת חדשה שבה כל האותיות הן אותיות גדולות.
- מחרוזות מתנהגות כמו משתנים בסיסיים. בזיכרון שמור עותק של כל ערך, וכאשר משנים את המחרוזת נוצרת מחרוזת חדשה.

י – אספן הזבל

ניתן לדייק ולומר כי אספן הזבל משחרר את הזיכרון כאשר אין גישה אליו מתוך התוכנית, למרות שלעיתים עדיין יש הפניה לעצם. למשל, בתרגיל הקודם יצרנו מעגל של עצמים. במעגל, לכל עצם יש הפניה מהעצם הקודם לו, כך שלכל עצם במעגל, קיימת הפנייה. אבל, אם אין שרשרת של הפניות שראשיתה במשתנה של התכנית וסופה בעצם במעגל, אזי המעגל אינו נגיש מהתכנית. אספן הזבל יודע לזהות גם מצבים כאלו, והוא ישחרר את כל העצמים שבמעגל.

4. דפי העבודה

פתרונות לכל דפי העבודה משולבים בדיסק למורה ומופיעים גם באתר המרכז להוראת המדעים, האוניברסיטה העברית בירושלים:

http://sites.huji.ac.il/science/unit4_2007/

דף עבודה מספר 1: הצבה של הפניות לעומת העתקה

מומלץ.

דף עבודה מספר 2: ניקח נקודות ונבנה מלבן

מומלץ.

דף עבודה מספר 3: נוסע ודרכון

רשות. ניתן לתת כשיעורי בית.

דף עבודה מספר 4: עצם מורכב

מומלץ. יכול להיפתר באופן תיאורטי לגמרי.

דף עבודה מספר 5: ספר טלפונים

חובה.

דף עבודה מספר 6: עצם שתכונתו מטיפוס עצמו

מומלץ כשיעורי בית.

דף עבודה מספר 1 – הצבה של הפניות לעומת העתקה

תרגול השמה של הפניות.

דגשים:

הבהרת ההבדל בין השמה של הפניות לבין העתקה. מטרת המשימה לתרגל תחילה שינוי של p1 דרך p2, ואחר כך העתקה של p1 ל-p3 ושינוי p3. ניתן להוסיף שאלת ביניים בדיון ולברר עם התלמידים מה לדעתם יחזירו ההשוואות:

```
p1 == p2; p1 == p3; p2 == p3
```

דף עבודה מספר 2 – ניקח נקודות ונבנה מלבן

תרגול בסיסי של עצם מורכב. הפעלת פעולותיה של תכונה שהיא עצם.

הייצוגים האפשריים הנוספים הם:

1. נקודה אחת, אורך ורוחב המלבן.
2. ארבעה משתנים מטיפוס **int**, המייצגים את שיעורי הנקודות (במקום שתי תכונות מטיפוס Point).

דגשים:

1. רצוי לדון עם התלמידים בייצוג בעזרת שתי נקודות, כפי שמוצע בתרגיל: מדוע מספיקות רק שתי נקודות? מדוע לא ארבע?
2. אחרי שמשנים את הייצוג, יש לציין כי הממשק לא השתנה (ולכן המחלקה מייצגת טיפוס נתונים מופשט).
3. בעיקר למורי הג׳אווה, מומלץ לדרוש את תיעוד המחלקות, כדי שהתלמידים יתרגלו לרעיון שמחלקה שהם כותבים מתועדת באופן מלא, כך שמכאן ואילך ניתן יהיה לעבוד איתה על פי הממשק בלבד (את הממשק ניתן להפיק בקלות כפי שכבר למדנו בעזרת ה-javadoc).

דף עבודה מספר 3 – נוסע ודרכון

תרגיל מסכם ומקיף.

תרגול יחס ההרכבה.

תרגול העתקה של עצם.

דגשים:

ניתן לפתח דיון בכיתה על הצורך בהעתקת עצם מטיפוס Date לעומת עצם מטיפוס Passport. כאשר יוצרים נוסע לא כדאי ליצור עותק של הדרכון, כיוון שאם מעדכנים את הדרכון במקום אחר, רצוי שהוא יתעדכן בכל המקומות שהוא מופיע (ולכן עדיף שיהיה עצם אחד של דרכון וכמה

הפניות יפנו אליו). כדי לבנות מערכת אמיתית כדאי להגביל את הגישה החופשית לדרכון ואת האפשרות לעדכנו.

דף עבודה מספר 4 – עצם מורכב

דף זה יכול להינתן בכיתה כבוחן, כיוון שפתרונו יכול להיעשות באופן תיאורטי לגמרי וכיוון שהוא מהווה תרגול ומוכיח הבנה של חלקים נרחבים מראשית הפרק.

דף עבודה מספר 5 – ספר טלפונים

דף זה הוא הראשון בסדרת מטלות המתייחסות לספר הטלפונים. זוהי הדוגמה שאליה יחזרו התלמידים בשיעורי הבית שוב ושוב, כאשר הם יידרשו לשנות ייצוגים ולשפר מימושים על פי התכנים שיילמדו בהמשך הפרק: שרשרת חוליות, רשימה ומפה. דוגמה נוספת נמצאת בגוף הפרקים והיא "המינהל הבית ספרי" וה-StudentList. חשוב מאוד שהתלמידים יבצעו את הדף בשלמותו, שכן חזרה עליו בכל שלב בהמשך תחדד את השיפורים והיכולות המתאפשרות עם התקדמות הלימוד ותדגים זאת באופן מוכר וברור.
ייצוג ספר הטלפונים בשלב זה ייעשה בעזרת מערך, דבר שידגים לתלמידים עד כמה מבנה נתונים זה אינו יעיל כאשר מבקשים לטפל באוסף דינמי.

דגשים:

1. ספר טלפונים אינו מוגבל בגודלו, ואכן כך הוא מתואר בנוסח השאלה. רצוי להשתחרר מהתחושה שמערך הוא מבנה מוגבל, שכן כאשר המערך מתמלא ניתן ליצור מערך חדש גדול יותר, להעתיק אליו את המערך הישן, ולהמשיך בעבודה עם המערך החדש. בספריות של השפות החדשות יש שימוש נרחב במערכים, ולרוב טיפוסי האוספים יש מימושים על ידי מערכים בנוסף למימושים בעזרת חוליות. לדוגמה, המחלקה הפופולרית בג'אווה ArrayList היא מימוש של רשימה המשתמש במערך. הבנה זו של האוסף תסייע לתלמידים בהמשך לימודיהם.
2. שלב המידול חשוב בשאלה זו, שכן קשה לבצע עם התלמידים מידול של מערכות גדולות, אך במערכת סגורה שכזו נוח וטוב לדון במחלקות הנדרשות לצורך ביצוע המשימה.
3. אם ברצונכם לחסוך בזמן הכתיבה של התלמידים, ניתן לתת להם את Contact כמחלקה קיימת שעליה תתבסס עבודתם (כמובן שרק לאחר שיזהו בשלב המידול שהם זקוקים לה).
4. הדיון, שנערך בפרק לגבי רשימת התלמידים והחזקתה ממוינת או לא, תקף גם לגבי ספר הטלפונים. כדאי להעיר את תשומת הלב של התלמידים לכך.

דף עבודה מספר 6 – עצם שתכונתו מטיפוס עצמו

תרגול מעשי של מבנים מקושרים.

הבנה בפועל של כל נושאי ההפניות שתוארו בפרק.

תרגיל זה הופך את כל התיאוריות והרעיונות שנלמדו על הפניות ופרמטרים ועל ערכי החזרה למובָנים. כאשר התלמידים נדרשים לעבוד איתם בפועל. חשוב מאוד שהתלמידים יבצעו תרגיל זה לפני שיתחילו ללמוד את הפרק על שרשרות של חוליות. ההתנסות תבהיר להם נושאים רבים שיעזרו להם בהמשך.

דף העבודה יכול להינתן כשיעורי בית שיידונו לאחר עשייתם – בכיתה.

סעיף א – מתרגל את המבנה הלינארי.

סעיף ב – מתרגל את המבנה ההיררכי.

דגשים:

1. אם החרוז הנשלח לפעולה הראשונה הוא החרוז האחרון, לא נקבל הדפסות כלל. אפשר לבקש הדפסה מתאימה שתסביר מה קרה.
2. רמז: למנהל החברה אין אחראי. החליטו אם אתם רוצים לתת את הרמז הזה לתלמידים בתחילת התרגיל או לתת להם לחשוב לבד.
3. בסעיף 5 אין צורך לפתור את הבעיה אלא להציע לה פתרון תיאורטי בלבד.