

פרק 9 רשימה

1. הזמן הנדרש

10 שעות.

2. מטרות

1. היכרות עם האוסף רשימה.
2. הבנת ההבדל בין מבנה נתונים וטיפוס נתונים מופשט בהקשר של טיפוס אוסף.

3. מבנה הפרק

מבוא כללי

• תפיסת המושג רשימה

אנו פותחים את הפרק במוטיבציה להגדרת רשימה כהמשך ישיר למחסנית ותור. במחסנית ובתור ההוצאה וההכנסה נעשו בקצוות. לעומת זאת ברשימה ההכנסה וההוצאה נעשות בכל **מקום** ולכל **מקום**, ונשאלת השאלה – מהו מקום? מה הגדרתו?

הגישה האינטואיטיבית להגדרת מקום היא לראותו כאינדקס, כלומר מקומו המספרי של האיבר. תלמידים נוטים באופן טבעי לגישה זו. חשוב להבין שזו אינה טעות וכי גישה זו קיימת גם בספרות האקדמית. בג'אווה ובסישרפ קיימת מחלקה בשם LinkedList המתייחסת למקום ברשימה כאינדקס, וכדי לשפר את יעילות הפעולות היא משתמשת ברעיון מופשט מתקדם הנקרא איטרטור (Iterator).

הגישה המוצגת בפרק היא גישה לפי **מקום בזיכרון** : **positional** – גישה זו מתייחסת למקום מסוים ברשימה, לא כאינדקס, אלא כהפניה לחוליה. אנו מציגים את מקומו של איבר ברשימה בעזרת המחלקה Node.

דרך אחרת היא לבנות אובייקט שאחראי על ניהול המקום בזמן הריצה (כך פועל האיטרטור, אך אנו לא נוכל להשתמש ברעיון זה לפני שהתלמיד יתקדם רבות בלימודיו). החיסרון של גישה זו הוא בכך שהיא אינה מאפשרת להגדיר את הרשימה כטיפוס נתונים מופשט:

- אופן הייצוג של מקום ברשימה חשוף למשתמש
 - בזמן שינוי הייצוג משתנה ממשק הרשימה
 - הרשימה חשופה לשינויים במבנה שלה באופן בלתי רצוי: דרך פעולות החוליה ניתן בקלות לפגוע בהגדרת הרשימה וליצור רשימה מעגלית למשל. הרשימה אינה יכולה להגן על עצמה משום שהחוליה שלה חשופה.
- סעיף י בפרק מסכם באופן ברור את הרעיון שרשימה היא **מבנה נתונים**, אך אינה יכולה להיחשב כ**טיפוס נתונים מופשט**. בכיתה ממוצעת אין צורך להתעמק בהגדרות המדויקות של

"טיפוס נתונים מופשט" לעומת "מבנה נתונים", אף שהדברים מובאים במפורש וברמיזות בפרקים. כיתות מתקדמות ומורים המחבבים את הנושא יכולים בהחלט להתעכב על הגדרות אלה. לטובת התעמקות זו שולבו תרגילים שונים בסוף הפרק העוסקים בהגדרת טיפוס נתונים מופשט ובמבני נתונים. כיתות שדילגו על ההבחנות יכולות לבצע את התרגילים מבלי להתעכב על פרטי ההבחנות בסעיפים הללו.

- ניתן לראות את הרשימה המשורשרת כמחלקה עוטפת לשרשרת חוליות. עד כה הכרנו רק את מחלקת החוליה. דיברנו על שרשרת חוליות כאוסף של חוליות, אך לא הגדרנו מחלקה עבור השרשרת. הרשימה היא בעצם מחלקה המנהלת את שרשרת החוליות: מכניסה ומוציאה את האיברים, ומאפשרת את הסריקה של השרשרת.

- הגדרת המחלקה עבור השרשרת פותרת גם חלק מהמגבלות שעליהן דיברנו בהקשר לשרשרת החוליות:

עבור שרשרת החוליות לא יכולנו להגדיר פעולות המקבלות אותה ומבצעות הכנסה או הוצאה למקום הראשון בשרשרת. זאת כיוון שפעולות אלה דורשות שינוי של ההפניה הראשונה בשרשרת החוליות. פעולה סטטית המקבלת את ההפניה לחוליה הראשונה בשרשרת החוליות אינה יכולה לשנות הפניה זו (כלומר אינה יכולה לגרום לה לפנות לעצם אחר). לעומת זאת ברשימה אין שום בעיה לעשות זאת, כיוון שפעולות ההכנסה וההוצאה הן פעולות ממשק, ולכן שינוי ההפניה הראשונה בשרשרת החוליות הוא שינוי התכונה של המחלקה ולא שינוי ההפניה שהתקבלה כפרמטר.

- כפי שראיתם במבוא ותראו בהמשך הפרק, אנו משתפים אתכם בסיבות לקבלת ההחלטות על ממשק הרשימה, כפי שהוא מוצג בפרק, וכן בגישות אלטרנטיביות שבהן החלטנו שלא לבחור, ומשתדלים להסביר את הטעמים לבחירותינו. נראה לנו שהבנת האילוצים ומערכת ההחלטות יכולה לעזור למורים בהתמודדות עם האילוץ של רשימה שאינה טיפוס נתונים מופשט, וכן בהתמודדות עם עוד שאלות שיכולות לעלות ולצוף בכיתות.

• רשימה – דוגמאות מהחיים

פעמים רבות נזקק מורה לדוגמה מהחיים כדי לשכנע את תלמידיו בנחיצות טיפוס האוסף החדש. לא כדאי להיצמד לדוגמה באופן מוחלט מכיוון שלרוב תימצא נקודת תורפה שבה הדוגמה אינה מתנהגת לחלוטין כמו רשימה כללית. אך די בדוגמאות כדי להבהיר את הנחיצות והטבעיות של סידור אוספים במבנה של רשימה. להלן מבחר דוגמאות שאספו משתתפי קורס מורים מובילים, שנת 2007.

הדוגמה הטובה ביותר לשימוש ברשימה, כאוסף דינמי שהגישה אליו אפשרית בכל מקום, היא סדרת מספרים שיש להוסיף לה או לגרוע ממנה איברים, ספר טלפונים או ספר כתובות ממוינים שבהם מכניסים ומוציאים פרטי אנשים על פי מקומם במיון. נתונה גם הדוגמה הדומה הבאה שאינה עוסקת ביישום ממוחשב שעיקרה פעולה פיזית:

סידור ספרים בספרייה, כאשר הם מקוטלגים וממוינים, נראה כרשימה חד-כיוונית (ממוינת!). כדי לקרוא בספר מסוים חייבים להגיע אליו ולשלוף אותו מהמדף. כדי להחזיר ספר למדף חייבים למצוא את מקומו המדויק והנכון ברצף הספרים.

- הדוגמאות הבאות הן מסגנון אחר. הן משקפות יותר את הקשר הסדרתי-הצבעתי שבין איברי הרשימה, אך הן רחוקות יותר מהיישום האמיתי שלה כמייצגת אוסף דינמי וגמיש:
1. מחפשים את המטמון או סימני דרך: שרשרת המשימות שחייבת להתבצע לפי הסדר (זהו למעשה סעיף כללי שמאחוריו מסתתרות תוכניות טלוויזיה ומרוצים שונים שפועלים לפי אותו העקרון).
 2. משחקי מחשב שבהם תחנות או שלבים וניקוד מצטבר. כמובן שאין צורך במשחק ממוחשב לצורך העניין, ודי בכל משחק המורכב משלבים שנבנים זה על גב זה (חוזר לרעיון של סעיף 1).
 3. שרשרת תגובות להודעה בפורום (טוקבקים) – ראש הרשימה היא ההודעה המקורית או המאמר שאליו מתייחסים ומכאן יש רשימה משורשרת של תגובות.
 4. מסלול נסיעה ברכבת או באוטובוס (אולי מורגש יותר במטרו, למי שמכיר): רצף וסדר התחנות נראה כרשימה משורשרת שיש להתקדם על פניה.

א – ייצוג הרשימה

ייצוג הרשימה על ידי שרשרת חוליות והגדרת "מקום" כהפניה לחוליה הופכים את הרשימה מרשימה כללית לרשימה משורשרת. כפי שאנו מדגישים בהמשך הפרק, רשימה משורשרת אינה טיפוס נתונים מופשט. הממשק שלה תלוי בייצוג שלה. (זאת כיוון שהפניה לחוליה מופיעה בממשק הן כפרמטר הן כערך החזרה).

ניתן לראות ברשימה משורשרת את ההתפתחות של **שרשרת החוליות** לכדי **מחלקה**. עד כה המחלקה היחידה שהוגדרה הייתה החוליה. לא הייתה מחלקה נפרדת שציינה את שרשרת החוליות. רשימה משורשרת היא מחלקה כזו. כל החסרונות שהיו לנו בשרשרת החוליות אינם קיימים ברשימה משורשרת:

- ניתן להעביר את הרשימה המשורשרת כפרמטר, ולבצע הכנסה והוצאה של האיבר הראשון.
- רשימה משורשרת יכולה להיות ריקה.

ב – דיון בפעולות הרשימה

- כיוון שהחוליה חשופה, הפעולות שלה נגישות, ולכן ניתן להשתמש בהן. כך הפעולה `getNext(...)` / `GetNext(...)` מסייעת לנו, כיוון שהיא מאפשרת להתקדם מחוליה אחת לחוליה הבאה, ומאפשרת סריקה. לעומת זאת, הפעולה `setNext(...)` / `SetNext(...)` עלולה לחבל במבנה הרשימה שכן היא מאפשרת הכנסה של חוליות לתוך הרשימה שלא דרך פעולת ההכנסה הפורמלית של הרשימה. באמצעות פעולה זו ניתן להרוס את המבנה הלינארי של הרשימה (למשל להפוך את הרשימה למעגלית). לכן הרשימה אינה טיפוס נתונים מופשט, כפי שאנו מבהירים גם בפרק עצמו.

ניתן היה להגדיר פעולה נוספת בממשק הרשימה, במקום להשתמש בפעולה של החוליה. פעולה זו הייתה מקבלת חוליה מסוימת ומחזירה את החוליה הזו:

```
public Node<T> getNext (Node<T> pos)           : בג'אווה:
public Node<T> GetNext (Node<T> pos)         : בסישרפ:
```

הבעיה היא שבמצב זה התלמיד יכול להשתמש בשתי פעולות להשגת אותה המטרה: הן בפעולת החוליה והן בפעולת הרשימה. כדי שלא תיווצר כפילות זו, בחרנו להשתמש בפעולה של החוליה, ולא ליצור פעולה נוספת ברשימה.

- בשונה מטיפוס המחסנית, אין לרשימה ממשק סגור ואחיד, המקובל בספרות. ניתן למצוא ממשקים שונים לרשימה. רוב הממשקים משתמשים בכלים מתקדמים שאין דרך להציגם ביחידת לימוד זו (למשל איטרטור).

ג – שימוש בפעולות הממשקים

בדומה למחסנית, אנו מתחילים בהצגת רשימה גנרית. בכיתות חלשות יותר, המתקשות עדיין עם נושא הגנריות, ניתן להציג תחילה רשימה שאינה גנרית, ורק אחר כך לעבור לרשימה גנרית.

ג.2. הכנסת ערך לרשימה

- לפי המפורט בפרק, ניתן להגדיר את פרמטר המקום שאותו מקבלת הרשימה באופנים שונים. ניתן להגדירו כמקום שהערך ייכנס אחריו, או מקום שהערך ייכנס לפניו. כדאי לדבר עם התלמידים על ההשלכות הנובעות מכל אחת מהבחירות. למשל, החלטה על הכנסה לפני המקום הנתון, מחייבת לחפש את המקום הקודם לנתון עבור כל הכנסה. כלומר: יעילות פעולת ההכנסה יורדת והפעולה הופכת יקרה יותר, שכן היא מסדר גודל: $O(n)$.
- חשוב לדבר על מקרי קצה, למשל: הכנסה למקום הראשון. ניתן להגדיר פעולת הכנסה פרטית למקרה זה, אך אנו בחרנו במקרה זה לזמן את הפעולה עם הערך `null`. במקום להגדיר מקום מיוחד ניתן להגדיר פעולה מיוחדת `insertFirst(...)` / `InsertFirst(...)`, שתטפל בהכנסת איבר למקום הראשון. החיסרון בפעולה מיוחדת להכנסת האיבר הראשון יתגלה במקרים שבהם עוברים על רשימה אחת ותוך כדי כך בונים רשימה אחרת על פי כלל מסוים. כאשר הפעולה להכנסת האיבר הראשון היא פעולה מיוחדת, אזי יש לסרוק תחילה את כל הרשימה ולבדוק האם התנאי מתקיים, ובשלב שני להכניס את האיבר הראשון. רק לאחר מכן יש לסרוק עוד פעם את כל הרשימה ולהכניס את יתר האיברים. בבחירה שלנו לעומת זאת אין צורך בכך, פעולת ההכנסה היא פעולה אחידה: ההכנסה למקום הראשון נראית בדיוק כמו כל הכנסה אחרת ויש רק פעולה אחת להכנסה בכל מקום.
- בגרסה הקודמת של יחידת הלימוד 'עיצוב תוכנה' הייתה קיימת חוליה ראשונה ריקה שנקראה "עוגן". זהו איבר דמה, והוא קודם לאיבר הראשון ברשימה. בכך הוא פותר את בעיית ההכנסה של האיבר הראשון לרשימה. חסרונו הבולט של העוגן, הוא בכך שהוא מהווה פתרון של בעיה מקומית טכנית שהלומד לא בהכרח הבין והפנים אותה. התוספת הנלווית לרשימה אינה מובנת ללומד לפני שהתעמק במימושים היותר מורכבים. לכן העדפנו לוותר על העוגן, להראות את הבעיות שעולות במימושים השונים, ולהציג את מימוש הרשימה עם העוגן כתרגיל בלבד.

- ערך ההחזרה של פעולת ההכנסה. ניתן היה שלא להחזיר ערך כלל, אך בחרנו להחזיר את המקום של החוליה שהוכנסה. הדבר נעשה מתוך שיקולי נוחות לצורך הסריקה. חשוב ללמד את התלמידים להשתמש בערך ההחזרה (בעזרת הדוגמאות המופיעות בפרק).
- דוגמה מעניינת: הכנסת איברים לרשימה (של מחרוזות):
בג'אוה:

```
List<String> lst = new List<String>();
Node<String> p = null;
for (int i =0; i<4; i++)
{
    //קלט של מחרוזת נוספת...
    p = lst.insert (p, str);
}
```

בסיסית:

```
List<string> lst = new List<string>();
Node<string> p = null;
for (int i =0; i<4; i++)
{
    //קלט של מחרוזת נוספת...
    p = lst.Insert (p, str);
}
```

כדאי לדון על אופן קידום ההפניה: כאשר משתמשים בערך החזרה של הפעולה, המחרוזות מתוספות לסוף הרשימה על פי סדר קליטתן. כאשר לא משתמשים בקידום ערכו של p , הפעולה אפשרית ותקינה אך משמעותה שונה: האיברים מתוספים לראש הרשימה!

ג.3. הוצאת ערך מרשימה

יכולנו להגדיר את פעולת ההוצאה בדומה לפעולת ההכנסה, כך שתבצע לאחר האיבר הנתון. בחירה זו עבור פעולת ההוצאה הייתה מצמצמת את יעילותה. פעולת ההוצאה תוגדר על המקום הנתון כיוון שכך היא נתפסת באופן אינטואיטיבי (בפעולת הוצאה ניתן להוציא את האיבר שפונים אליו, אבל בהכנסה אי אפשר להכניס את האיבר החדש במקום שפונים אליו, כי במקום זה קיים איבר). ניתן להסביר לתלמידים שקיימת א-סימטריה בין פעולות ההוצאה וההכנסה.

ג.4. סריקה

- לעתים תנאי העצירה לא יהיה הבדיקה של ה- pos , אלא בדיקה של האיבר העוקב:
 $pos.getNext() / pos.GetNext()$
- למשל, כאשר רוצים למחוק את איבר המקיים תנאי מסוים: אם נבדוק את התנאי על האיבר עצמו, הרי לאחר שנמצא את האיבר, נצטרך לסרוק שנית את שרשרת החוליות כדי לבצע את המחיקה. לעומת זאת אם נבדוק בכל שלב אם האיבר הבא מקיים את התנאי, בעת מציאת האיבר נעמוד במקום אחד לפניו ונוכל למחוק אותו ללא סריקה נוספת.
- יש לשים לב שהתלמידים אינם מתבלבלים בין:

בג'אווה: `p = p.getNext()`, לבין `p.setNext(p.getNext())`
 בסישרפ: `p = p.GetNext()`, לבין `p.SetNext(p.GetNext())`

אם טעות שכזו עולה בכיתה, כדאי להפנות אליה את תשומת הלב של כלל הכיתה, לצייר על הלוח ולהסביר את ההבדל ואת משמעותו.

תשובות לשאלות המחשבה המופיעות בסעיף זה:

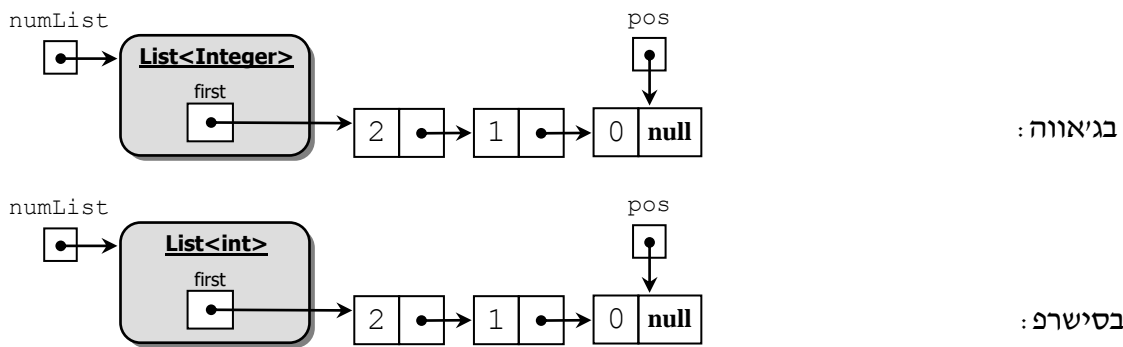
? תארו את הרשימה המתקבלת מהפעלת הקוד שלפניכם על רשימת המספרים הריקה lst:

בג'אווה: `for (int i=0; i<3; i++) lst.insert(null, i);`

בסישרפ: `for (int i=0; i<3; i++) lst.Insert(null, i);`

תשובה:

בשאלה זו ההכנסה מתבצעת כל הזמן למקום הראשון ברשימה. לכן, הרשימה המתקבלת תהיה:



בג'אווה:

בסישרפ:

? האם ניתן לכתוב את הפעולה `getPosition(...)` / `GetPosition(...)` כפעולה פנימית?

תשובה:

לא, כיוון שבפעולה זו אנו מבצעים השוואה לגבי ערכו של הפרמטר, ולא כל טיפוס ניתן להשוואה.

ד – כתיבת המחלקה רשימה

אף שהרשימה היא גנרית והטיפוס שלה הוא טיפוס כללי T, אנחנו יכולים להגדיר את הפעולה `toString() / ToString()` כפעולה פנימית של הרשימה. זאת כיוון שבשפות התכנות ג'אווה וסישרפ קיימת הנחה בסיסית כי לכל טיפוס מוגדרת פעולת `toString() / ToString()` (בעצם כל טיפוס יורש אותה מטיפוס האב שלו Object).

ניתן לממש את הפעולה בשני אופנים :

באופן איטרטיבי : לסרוק את הרשימה ולהדפיס את החוליות שלה.

באופן רקורסיבי : הפעולה תפעיל פעולת עזר, ובזימון היא תשלח את חוליית הראש של שרשרת החוליות. פעולת עזר תקבל חוליה ותדפיס את ה-info ואת ה-next של החוליה.

תשובה לשאלת המחשבה המופיעה בסעיף זה :

? השלימו את קוד הפעולה toString() / ToString().

סריקה איטרטיבית:

בג'אוה :

```
public String toString()
{
    String str = "[";
    Node<T> pos = this.first;
    while (pos != null)
    {
        str = str + pos.getInfo().toString();
        if (pos.getNext() != null)
            str = str + ",";
        pos = pos.getNext();
    }
    str = str + "]";

    return str;
}
```

בסישרפ:

```
public string override ToString()
{
    string str = "[";
    Node<T> pos = this.first;
    while (pos != null)
    {
        str = str + pos.GetInfo().ToString();
        if (pos.GetNext() != null)
            str = str + ",";
        pos = pos.GetNext();
    }
    str = str + "]";

    return str;
}
```

פתרון רקורסיבי:

בג'אווה:

```

public String toString()
{
    String str = "[";

    if (this.first != null)
        str = str + helptoString(this.first);
    str = str + "]";

    return str;
}

public String helptoString(Node<T> node)
{
    String str = node.getInfo().toString();

    if (node.getNext() != null)
        str = str + "," + helptoString(node.getNext());

    return str;
}

```

בסישרפ:

```

public string override ToString()
{
    string str = "[";

    if (this.first != null)
        str = str + HelpToString(this.first);
    str = str + "]";

    return str;
}

public string HelpToString(Node<T> node)
{
    string str = node.GetInfo().ToString();

    if (node.GetNext() != null)
        str = str + "," + HelpToString(node.GetNext());

    return str;
}

```

ה – יעילות פעולות הממשק

- יעילות פעולת ההוצאה היא $O(n)$, משום שכדי להוציא איבר ממקום מסוים, יש למצוא את האיבר הקודם לו. חיפוש האיבר הקודם מתבצע על ידי סריקת הרשימה בעזרת הפנייה, החל בראשה. בשלב הזה כדאי לעודד את התלמידים לחשוב איך לייעל את הפעולה על ידי שינוי הייצוג (לקראת התרגיל של רשימה דו-כיוונית).

- בניתוח יעילות הפעולה `remove(...)` / `Remove(...)` חשוב לשים לב לעניין עדין: אם מבצעים את ההוצאה של האיבר הראשון אזי אין צורך לחפש את האיבר הקודם לו, ולכן יעילות הפעולה היא $O(1)$. הבנה של עיקרון זה חשובה למשל במקרה שבו מייצגים מחסנית באמצעות רשימה (כפי שנעשה בפרק). במימוש זה, יעילות פעולת ההוצאה היא $O(1)$ ולא $O(n)$, כיוון שתמיד מבצעים את ההוצאה של האיבר הראשון. שלב זה עלול לבלבל מאוד את התלמידים, כיוון שהם רגילים לניתוח על פי המקרה הגרוע, ולכאורה בפעולת הוצאה המקרה הגרוע הוא הוצאה של האיבר שאינו ראשון. אבל: ניתוח על פי המקרה הגרוע מתאים כאשר איננו יודעים איזה מקרה של הוצאה יזומן. במקרה בו אנו יודעים בוודאות שתמיד לא מזמנים את המקרה הגרוע (כמו למשל הוצאה עקבית של האיבר הראשון בלבד), אין שום סיבה לחשב את היעילות על פי המקרה הגרוע.

1 – פעולות נוספות על רשימה

- משום חשיבותם של כמה דברים נזכרים שוב:
 1. פעולות פנימיות וחיזוניות. פעולות פנימיות אינן יכולות לבצע חישובים והשוואות על הערכים השמורים ברשימה הגנרית. בין עצמים שונים יכולים להיות הבדלים בהגדרת פעולת השוויון, בהגדרת פעולת ה-`compareTo(...)` / `CompareTo(...)` ובהגדרת פעולת הסכימה (+) וכד'. לכן פעולות המצריכות התייחסות לערכי הרשימה יוגדרו מראש כפעולות חיזוניות על רשימות קונקרטיות. יוצאת דופן היא הפעולה `toString()` / `ToString()` – ראו בפירוט לעיל.
 2. כל הפעולות החיזוניות על רשימה ללא יוצא מהכלל יוגדרו על רשימות קונקרטיות ולא יוגדרו כגנריות, גם אם הן אינן מתייחסות לערכי הרשימה, לדוגמה: הפעולה אורך רשימה. הסיבה להחלטה זו: א) שמירה על קו פעילות אחיד תמנע הסתבכות של התלמידים; ב. לפעולות חיזוניות גנריות יש תחביר מיוחד:


```
public static <T> int  getLength (List<T> lst)           בג'אווה:
public static <T> int  GetLength (List<T> lst)         בסישרפ:
```
- כדי לא להיכנס לתסבוכות מיותרות, אנו משתמשים רק בחלק קטן של המנגנון הגנרי ומוותרים על כל השאר.

בכיתות חזקות כדאי לדון בכך שניתן להגדיר פעולות כגון `getLength(...)` / `GetLength(...)` כפעולות פנימיות או חיזוניות של רשימה. לעומת זאת, לא הגיוני להגדיר פעולה כזאת כפעולה פנימית של החוליה. זאת כיוון שהחוליה אינה מכירה את כל הרשימה, אלא רק את עצמה. לכן לא הגיוני להגדיר בחוליה פעולה כזאת.
- 3. לגבי המשמעות של מקום ברשימה. כאמור, יש שתי אפשרויות סבירות לייצוגו של המושג מקום:
 - אינדקס מספרי (אפשרי הן בייצוג על ידי מערך והן בייצוג על ידי שרשרת חוליות).
 - הפניה לחוליה (כאשר הייצוג נעשה על ידי שרשרת חוליות).
 אנו בחרנו באפשרות השנייה מבין אלה.

יש הבדל משמעותי בין שני הייצוגים אלה. נניח לדוגמה כי משתנה חיצוני pos מחזיק מקום ברשימה: לפי האפשרות הראשונה, הוא מחזיק את האינדקס 5; ולפי האפשרות השנייה, הוא מחזיק הפניה לחוליה החמישית. נניח עוד כי זומנה פעולה שהוסיפה איבר לרשימה, אחרי המקום הראשון. עבור הייצוג של מקום בעזרת אינדקס, הערך שבמשתנה pos (שהוא עדיין 5) מצביע עכשיו על איבר שונה מזה שהצביע עליו קודם, כיוון שהאיבר שהיה קודם הרביעי ברשימה הפך עכשיו להיות החמישי. אם המתכנת יתעלם משינוי זה, סביר שתגרם שגיאה בזמן ריצת התוכנית. לעומת זאת, אם המשתנה pos מחזיק הפניה לחוליה, אזי הפניה זו ממשיכה להפנות לאותה חוליה, והעובדה שנוספה חוליה לפנייה אינה משנה זאת כלל. הפעולה insertionSortInPlace(...) / InsertionSortInPlace(...) (סעיף ו) מדגימה עניין זה. בפעולה זו, המשתנה untilPos מכיל הפניה לאיבר הראשון שעדיין לא הוכנס לקטע הממוין. כאשר מכניסים לקטע הממוין איבר זה בחוליה חדשה, אין זה משפיע על ההפניה שבמשתנה זה. לעומת זאת, אילו המקום ברשימה היה מיוצג על ידי אינדקס מספרי, אזי לשם ביצוע ההכנסה, התוכנית הייתה צריכה לא רק לבצע הזזה של איברים בקטע של המערך, אלא גם להגדיל את ערכו של האינדקס במשתנה זה ב-1. בפעולה זו, דבר זה אפשרי, כי pos הוא משתנה פנימי של הפעולה. כאשר ייתכן שיש משתנים אחרים, מחוץ לפעולה, המחזיקים אינדקסים, אין לבעיה פתרון כללי סביר.

- הנקודות החשובות להדגשה בדוגמאות המוצגות בסעיף:

1. יצירת תת-רשימה ובה המספרים הזוגיים מרשימה נתונה

בדוגמה זו אנו רואים סריקה של שתי רשימות (ולכן צריך שתי הפניות). אחת הרשימות נבנית תוך כדי סריקה. יש לשים לב כי בהתחלה הרשימה החדשה ריקה. לכן אחרי זימון השורה:

```
Node<Integer> pos2 = evenList.getFirst();           : בג'אווה
Node<int> pos2 = evenList.GetFirst();             : בס'שרפ:
```

ערכה של ההפניה pos2 הוא null. אם היינו מנסים לקדם אותה כפי שאנחנו מקדמים את pos1, אזי הייתה בג'אווה חריגה מטיפוס NullPointerException, ובס'שרפ חריגה מטיפוס NullPointerException. לכן, קידום של הפניה הסורקת רשימה' הנבנית תוך כדי הפעולה, נעשה באמצעות פעולת ההכנסה עצמה:

```
pos2 = evenList.insert(pos2, pos1.getInfo());     : בג'אווה
pos2 = evenList.Insert(pos2, pos1.GetInfo());     : בס'שרפ:
```

זו נקודה קצת קשה להבנה, ולכן מומלץ לתת לתלמידים להתמודד לבד עם התרגיל. רק אחר כך יש לסקור יחד פתרון לא נכון של הבעיה ולהראות שנגרמת חריגה.

ניתן להיעזר בשאלה מכוונת: נסו להעריך מה יקרה אם נקדם את שתי ההפניות באותו אופן:
בג'אוה:

```
if ((pos1.getInfo() % 2) == 0)
{
    evenList.insert(pos2, pos1.getInfo());
    pos2 = pos2.getNext();
}
pos1 = pos1.getNext();
```

בסישרפ:

```
if ((pos1.GetInfo() % 2) == 0)
{
    evenList.Insert(pos2, pos1.GetInfo());
    pos2 = pos2.GetNext();
}
pos1 = pos1.GetNext();
```

2. הכנסת ערכים לרשימה ממוינת

בתרגיל זה כדאי לדון עם התלמידים בדרך הפתרון לפני שמציגים אותו. ישנן כמה דרכים להתמודד עם הכנסה לרשימה ממוינת. דרך אחת: ביצוע שתי סריקות, אחת כדי למצוא את האיבר הראשון שגדול יותר מהאיבר שמוכנס, וסריקה שנייה כדי למצוא את האיבר הקודם לו (זאת כיוון שפעולת הכנסה חייבת לקבל את המקום הקודם למקום המוכנס). דרך אחרת: ביצוע סריקה עם שתי הפניות (זהו הפתרון המוצג בפרק). זהו פתרון שנוח מבחינה טכנית. דרך אפשרית נוספת היא לסרוק את הרשימה תוך כדי ביצוע הבדיקה של הערך הבא, ולא הערך הנוכחי. פתרון זה פחות נוח כיוון שבתחילת הסריקה (בחוליה הראשונה) איננו יכולים לבדוק את הערך הבא. בכיתה חזקה ניתן לתת לתלמידים להתמודד לבד עם האפשרויות השונות ולסקור את דרכי הפתרון בכיתה.

3. מיון רשימה באמצעות מיון הכנסה (insertion sort)

מיון במקום הוא קשה יותר, וניתן לוותר עליו בכיתות החלשות. בכיתות החזקות, מומלץ להציג בפני התלמידים את הרעיון של חלוקת הרשימה לשני חלקים: החלק הממוין והחלק הלא ממוין, ולתת להם להתמודד לבד עם הפתרון.

תשובות לשאלות המחשבה המופיעות בסעיף זה:

- א. הראו כי הפעולה תפעל כראוי בכל מקרי הקצה הקיימים.
ב. אילו שינויים יש לעשות בפעולה כך שתבצע את משימתה על רשימה של מספרים שלמים?

תשובה:

א. נבחן את מקרי הקצה: מצב של רשימה ריקה ופעולות על המקום הראשון והאחרון.
נסתכל בקוד הפעולה:
בג'אוה:

```
public static void insertIntoSortedList (List<String> lst,
                                         String str)
{
    Node<String> prev = null;
    Node<String> pos = lst.getFirst();

    while ((pos != null) && (pos.getInfo().compareTo(str)<0))
    {
        prev = pos;
        pos = pos.getNext();
    }

    lst.insert(prev, str);
}
```

בסישרפ:

```
public static void InsertIntoSortedList (List<string> lst,
                                         string str)
{
    Node<string> prev = null;
    Node<string> pos = lst.GetFirst();

    while ((pos != null) && (pos.GetInfo().CompareTo (str)<0))
    {
        prev = pos;
        pos = pos.GetNext();
    }

    lst.Insert(prev, str);
}
```

רשימה ריקה: ערכו של pos הוא null, לולאת ה-while כלל לא תתבצע ותזומן השורה האחרונה:

בג'אוה: lst.insert (prev, str);
בסישרפ: lst.Insert (prev, str);

כיוון שגם prev שווה null, האיבר יוכנס למקום הראשון כנדרש.
הכנסה למקום הראשון: אם האיבר להכנסה קטן מהאיבר הראשון, התוכנית לא תבצע את הלולאה, כי התנאי השני (ההשוואה בין ערך האיבר במקום הראשון והערך להכנסה) אינו

מתקיים. גם במקרה זה בדומה לקודם, נזמן את פעולת ההכנסה שבשורה האחרונה כאשר הפרמטר הראשון שווה `null`, ולכן נכניס למקום הראשון כנדרש. **הכנסה למקום האחרון**: אם האיבר להכנסה הוא הגדול ביותר, הלולאה תרוץ עד ש-`pos` יהיה שווה `null`, ו-`prev` יפנה לחוליה האחרונה בשרשרת. במקרה זה פעולת ההכנסה תכניס את האיבר למקום האחרון כנדרש.

ב. השינויים הנדרשים להפיכת הפעולה לנכונה עבור רשימה של מספרים שלמים: בג'אווה: בכל מקום שכתוב `String`, יש לכתוב `Integer`. את הפעולה `compareTo(...)` ניתן להשאיר (אפשר גם להחליפה לסימן `==`, אך אין בכך כל צורך). בסישרפ: בכל מקום שכתוב `string`, יש לכתוב `int`. את הפעולה `compareTo(...)` ניתן להשאיר (אפשר גם להחליפה לסימן `==`, אך אין בכך כל צורך).

? בפעולה `insertIntoSortedList(...)` / `InsertIntoSortedList(...)` (דוגמה 2) הפרמטר השני הוא ערך, ואילו כאן, בפעולה `insertIntoPartialSortedList(...)` / `InsertIntoPartialSortedList(...)` הפרמטר השני הוא חוליה. הסבירו מדוע בפעולה `insertIntoPartialSortedList(...)` / `InsertIntoPartialSortedList(...)` יש צורך בפרמטר מטיפוס חוליה.

תשובה:

בפעולה `insertIntoSortedList(...)` / `InsertIntoSortedList(...)` אנו מעוניינים רק בערך של האיבר להכנסה. פעולת ההכנסה של רשימה, שהיא זו המבצעת את ההכנסה בפועל, יוצרת חוליה חדשה עבור הערך. לעומת זאת, ההפניה `untilPos` מחזיקה מידע כפול: גם את הערך שאנו מעוניינים להכניס (בעזרת פעולת ההכנסה של רשימה) וגם את המקום שעד אליו הרשימה ממוינת. לכן חשוב להעביר את כל ההפניה ולנצל את כל המידע שהיא מכילה.

ז – ייצוג אוספים באמצעות רשימה

- בעבר ייצגנו אוספים באמצעות שרשרת חוליות ומערך. ייתכן שהתלמידים יחשבו שמשונה לייצג אוסף אחר באמצעות רשימה (המיוצגת בעצמה בעזרת שרשרת חוליות). יש לדון בכך שניתן להשתמש באוספים שבנינו כדי לייצג אוספים אחרים. בכיתות חזקות ניתן לתת לתלמידים להתמודד לבד עם כתיבת המחלקות.
- רצוי לדון עם התלמידים בייצוג התור בעזרת רשימה: בתור אנו צריכים גישה לשתי הקצוות. לתחילת הרשימה קל לגשת, אך לצורך גישה לסוף הרשימה עלינו לסרוק את כל התור. כדאי לדון עם התלמידים באפשרות שיפור: האם כדאי להוציא איברים מתחילת הרשימה ולהכניס איברים לסופה או להפך (שימו לב, אם בוחרים להוסיף תכונה המפנה לחוליה האחרונה, אזי הבחירה להוציא מסוף הרשימה בעייתית מאוד,

שהרי נצטרך לעדכן את ההפניה לחוליה האחרונה, וכדי לעשות זאת נצטרך לסרוק את כל הרשימה).

תשובות לשאלות המחשבה המופיעות בסעיף זה :

? ממשו את המחלקה כך שפעולת ההוספה Add(...) תוסיף את התלמיד למקומו הנכון לפי הסדר האלפביתי.
בג'אוה :

```
public void add(Student st)
{
    Node<Student> prev = null;
    Node<Student> pos = lst.getFirst();

    while ((pos != null) &&
           (pos.getInfo().getName().compareTo(st.getName())<0))
    {
        prev = pos;
        pos = pos.getNext();
    }

    lst.insert(prev, str);
}
```

בסישרפ :

```
public void Add(Student st)
{
    Node<Student> prev = null;
    Node<Student> pos = lst.GetFirst();

    while ((pos != null) &&
           (pos.GetInfo().GetName().CompareTo(st.GetName())<0))
    {
        prev = pos;
        pos = pos.GetNext();
    }

    lst.Insert(prev, str);
}
```

? כתבו את פעולת העזר המתאימה, הסורקת את הרשימה ומחזירה את המקום האחרון בה.

תשובה:

בג'אוה:

```
private static Node<T> getLast(LinkedList<T> list)
{
    Node<T> pos = list.getFirst() ;
    while (pos.getNext() != null)
    {
        pos = pos.getNext() ;
    }
    return pos ;
}
```

בסישרפ:

```
private static Node<T> GetLast(LinkedList<T> list)
{
    Node<T> pos = list.GetFirst() ;
    while (pos.GetNext() != null)
    {
        pos = pos.GetNext() ;
    }
    return pos ;
}
```

ח – מימוש רקורסיבי של פעולות ברשימה

תשובות לשאלות המחשבה המופיעות בסעיף זה:

? הפעולה שלפניכם מומשה בסעיף ג.4. דוגמה 2.

בג'אוה:

```
public static Node<Integer> getPosition(List<Integer> lst, int x)
```

בסישרפ:

```
public static Node<int> GetPosition(List<int> lst, int x)
```

כתבו את הפעולה כפעולה רקורסיבית (מותר להגדיר פעולות עזר רקורסיביות).

:תשובה

:בג'אווה

:נגדיר פעולת עזר:

```
private static Node<Integer> getPositionHelp(Node<Integer> node,
                                             int x)
{
    if (node.getInfo() == x)
        return node;
    else
    {
        if (node.getNext() == null)
            return null;
        else
            return getPositionHelp(node.getNext(), x);
    }
}
```

:הפעולה של הרשימה

```
public static Node<Integer> getPosition(List<Integer> lst, int x)
{
    if (lst.getFirst() == null)
        return null;
    else
        return getPositionHelp(lst.getFirst(), x);
}
```

:בסישרפ

:נגדיר פעולת עזר:

```
private static Node<int> GetPositionHelp(Node<int> node, int x)
{
    if (node.GetInfo() == x)
        return node;
    else
    {
        if (node.GetNext() == null)
            return null;
        else
            return GetPositionHelp(node.GetNext(), x);
    }
}
```

:הפעולה של הרשימה

```
public static Node<int> GetPosition(List<int> lst, int x)
{
    if (lst.GetFirst() == null)
        return null;
    else
        return GetPositionHelp(lst.GetFirst(), x);
}
```


ט. רשימה דו-כיוונית

תשובה לשאלת המחשבה המופיעה בסעיף זה:

? איך תשתנה היעילות של הפעולה `insertIntoSortedList(...)` / `InsertIntoSortedList(...)` שראינו

בדוגמה 2, בסעיף ו, כאשר הרשימה תיוצג באמצעות שרשרת חוליות דו-כיוונית? הסבירו.

תשובה:

יעילות הפעולה לא תשתנה. גם כאשר הייצוג יהיה בעזרת שרשרת חוליות דו-כיוונית, יעילות

הפעולה תהיה $O(n)$ (אם נוותר על המשתנה `prev` השיפור ביעילות יהיה לא משמעותי).

י – הרשימה – טיפוס נתונים מופשט?

תשובה לשאלת המחשבה המופיעה בסעיף זה:

? בדקו מה יתקבל בהדפסה? מה מקור הבעיה?

תשובה:

התוצאה היא הרשימה: `lst = [2, 3, 5]`. הבעיה היא ששתי השורות האחרונות בקוד אמורות

להתבצע בסדר הפוך, כך:

בג'אווה:

```
n.setNext(pos.getNext());  
pos.setNext(n);
```

בסירפ:

```
n.SetNext(pos.GetNext());  
pos.SetNext(n);
```

אם השורות מתבצעות בסדר המקורי שבו הן מופיעות בפרק, הרי כפי שקרה ליעקב, ברגע שמשנים את ההפניה לחוליה העוקבת ל-`pos` כך שתפנה לחוליה החדשה, מאבדים את כל שארית השרשרת, ואי אפשר לעדכן כראוי את ה-`next` של החוליה החדשה.

4. תרגילים

פתרונות לכל התרגילים מופיעים באתר המרכז להוראת המדעים, האוניברסיטה העברית בירושלים, ונגישים רק לציבור המורים:

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

שאלות 1–6: מעקב אחר פעולות הרשימה

מומלץ לפתור חלק מהשאלות בכיתה וחלק לתת כשיעורי בית.

שאלות 7–14: כתיבה של פעולות פנימיות וחיזוניות על רשימה

מומלץ לפתור חלק מהשאלות בכיתה וחלק לתת כשיעורי בית.

שאלה 15: הוספת הפעולה הבונה ל-StudentList

מומלץ לפתור בכיתה.

יש לפתח דיון עם התלמידים על אופן ההעתקה (העתקה עמוקה או שטחית), ולכוון אותם להוסיף פעולה בונה גם למחלקות Student ו-Parent.

שאלה 16: מימוש רשימה דו-כיוונית

מומלץ להתחיל בכיתה, לתת לתלמידים לסיים כשיעורי בית ולבדוק ביחד בכיתה.

שאלה 17: zip ו-unzip על רשימה

מומלץ להתחיל בכיתה ולתת לסיים בבית.

שאלות 18–21: מימוש טיפוס נתונים מופשטים

סוג תרגילים זה ממחיש את הרעיון של טיפוס נתונים מופשט. ניתן לדון עם התלמידים בשאלה האם שינוי המימוש משנה את הממשק או שאינו משנה את הממשק. כדאי מאוד לבצע לפחות חלק מתרגילים אלה. אותם טיפוסים ניתנים לייצוג גם באמצעות עצים, ואופני הייצוג השונים מדגישים את החשיבות של בחירת הייצוג בהתחשב ביעילות.

המלצה כללית – להוסיף לכל אחת מהשאלות על הטיפוסים שני סעיפים לדיון:

- i. האם הטיפוס הוא טיפוס נתונים מופשט?
- ii. האם ניתן להכליל את הטיפוס לטיפוס גנרי במקום שיהיה בעל אפיון מסוים?

שאלה 18 – קבוצה

לא ניתן לממש קבוצה גנרית באמצעות הכלים הקיימים ברשותנו כרגע, כיוון שבמחלקה זו דרוש שימוש בפעולת ההשוואה, ואין לנו כלים להגדיר מחלקה רק עבור טיפוס הניתן להשוואה. אם הזמן קצר, ניתן לוותר על מימוש פעולות האיחוד והחיתוך. אם מחליטים לממש פעולות אלה יש להסביר אותן היטב בכיתה ולהיעזר בדוגמאות.

קיימות כמה דרכים לממש את הקבוצה: באמצעות חוליה, באמצעות רשימה או באמצעות מערך. מימוש הקבוצה באמצעות חוליה או באמצעות רשימה אינו משנה את היעילות: ההבדל ביניהם הוא רק בנוחות כתיבת הפעולות.

במימוש בעזרת מערך כיוון שאין שום סדר בין איברי הקבוצה, ניתן למחוק את האיברים של הקבוצה באופן שישארו "חורים" במערך, למשל על ידי סימון תאים פנויים באמצעות ערך בוליאני כלשהו. במקרה כזה ההכנסה לקבוצה תתבצע תמיד ב- $O(n)$.

שאלה 19 – אוסף ממוין

שאלה זו מכילה נקודות התייחסות מעניינות רבות, ולכן מומלץ ביותר לבצע אותה ביחד בכיתה תוך דיון וליבון כל הנקודות הבאות:

אוסף ממוין לא ניתן לממש כאוסף גנרי, כיוון שעל ערכי האוסף חייבת להיות מוגדרת פעולת ההשוואה. ניתן לדון במימושים שונים של האוסף הממוין: חוליה, רשימה או מערך. בעתיד נדבר על מימוש נוסף מתקדם.

יש לשים לב שאין שום חיוב לשמור את הנתונים של האוסף באופן ממוין. למשתמש החיצוני אין דרך לדעת כיצד שמורים הנתונים והאם הסדר ביניהם זוכה לייצוג פנימי או בא לידי ביטוי רק בתוצאת הפעולות המתאימות (`getAll()` / `GetAll()` למשל). זוהי אחת התוצאות של היות האוסף טיפוס נתונים מופשט. בכל זאת, על ידי הטלת הגבלות שונות בנושא היעילות אנו יכולים לכוון את המתכנת לדרך ייצוג מסוימת.

באשר ליעילות: פעולת ההכנסה פחות נוחה במערך מאשר ברשימה או בשרשרת חוליות, כיוון שצריך להזיז את כל איברי המערך כדי להכניס איבר למקום מסוים. יעילות פעולה זו היא $O(n)$. לעומת זאת במערך, הפעולה `exists(...)` / `Exists(...)` יעילה, כיוון שניתן לבצע אותה ב- $O(\log n)$ על ידי ביצוע חיפוש בינרי (חציית המערך בכל פעם). ברשימה לעומת זאת, ההכנסה של האיבר היא ב- $O(n)$, כיוון שצריך לחפש את המקום הנכון להכנסה, וגם הפעולה `exists(...)` / `Exists(...)` היא ב- $O(n)$, כיוון שאין לנו ידע לגבי אורך הרשימה. אם מחזיקים כתכונה את גודל הרשימה, ניתן לבצע חיפוש בינרי על הרשימה בדומה למערך.

פעולת המחיקה אף היא ב- $O(n)$, כיוון שמציאת האיבר היא ב- $O(n)$.

עצם ריק: הפעולה `getAll()` / `GetAll()` על אוסף ממוין ריק תחזיר עצם מטיפוס מערך, אך העצם יהיה ריק. אם נשאל לגבי אורכו של המערך המוחזר – נקבל 0. כדאי לחדד עם התלמידים את רעיון שקיים עצם מטיפוס מערך אף שהוא ריק, ולהדגים להם.

שאלה 20 – ספר הטלפונים

חזרה נוספת למשימת שיעורי הבית המלווה את היחידה, והפעם הייצוג יהיה בעזרת רשימה.

שאלה 21 – שאלה מבחינת בגרות

זוהי דוגמה לשאלת בגרות שהותאמה לרוח היחידה המחודשת. נביא את המקור ונבחן מה הצריך שינוי והתאמה. בחינה שכזו תעזור לכם בבואכם להחליף ולהתאים שאלות ישנות כך שיתאימו ליחידה.

נוסח שאלת הבחינה המקורית:

מדעי המחשב ב', קיץ תשס"ה, מס' 899205, 603

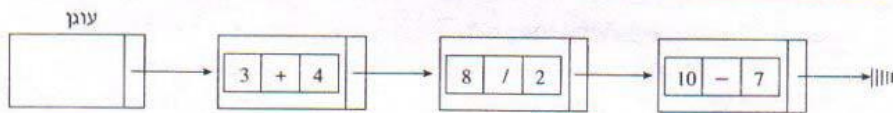
- 4 -

2. רשימה "חשבונית" L היא רשימה ששדה התוכן שלה מייצג ביטוי חשבוני.

הביטוי החשבוני מורכב משלושה חלקים:

- מספר שלם גדול מאפס
- תו אחד מבין ארבעת התווים:
 - + המייצג חיבור
 - המייצג חיסור
 - * המייצג כפל
 - / המייצג חילוק
- מספר שלם גדול מאפס

דוגמה לרשימה "חשבונית" L:



א. הגדר בסביבת העבודה את טיפוס שדה התוכן של איבר ברשימה "חשבונית" L.

ב. ממש בסביבת העבודה תת-תכנית calculate, שתקבל רשימה "חשבונית" L ומקום p ברשימה. p הוא מקום ב־ L שאינו סוף-רשימה ואינו עוגן-רשימה. התת-תכנית תחזיר את התוצאה המתקבלת מהביטוי החשבוני הנמצא באיבר שבמקום p.

ג. ממש בסביבת העבודה תת-תכנית sumExpressions, שתקבל רשימה "חשבונית" L, ותחזיר את הסכום הכולל של תוצאות הביטויים החשבוניים הנמצאים ברשימה. בעבור רשימה ריקה יוחזר 0. עליך להשתמש בתת-תכנית calculate.

בעבור הרשימה "החשבונית" L בדוגמה התונה, התת-תכנית sumExpressions תחזיר 14.

הערה: אין צורך לממש בסביבת העבודה את הפעולות של ממשק רשימה.

/המשך בעמוד 5/

ניתוח הנוסח המקורי:

הנוסח המקורי מבקש למעשה הגדרת פעולות חיזוניות (תת-תוכנית... שתקבל רשימה...). זוהי תוצאה של התכנות הפרוצדורלי שבו לא היו עצמים מטיפוסים שונים. כאשר ממדלים את הבעיה המתוארת בשאלה, בגישה מבוססת עצמים, מגלים שיש צורך להגדיר שני טיפוסים:

האחד הוא: 'ביטוי חשבוני' שהוא למעשה "טיפוס שדה התוכן" המתבקש בסעיף א. כיוון ש'ביטוי חשבוני' הוא עצם, אזי הפעולה המתוארת בסעיף ב, היא אחת מפעולות הממשק של טיפוס זה. הטיפוס הנוסף שצריך להגדיר הוא 'רשימה חשבונית', שאיבריה הם ביטויים חשבוניים. מכיוון שבגישה מבוססת עצמים מדובר בשאלה זו על שתי מחלקות שונות, יש מקום לברר האם התלמיד יכול למדל נכון ולשייך את הפעולות למחלקות המתאימות (יש לשים לב ששאלת מידול פתוחה היא שאלה קשה מדי לצורך שאלת מבחן, אך שאלת מידול המשייכת פעולות למחלקות מוגדרות היא שאלה סבירה בהחלט). תוספת נימוק של התלמיד תעזור לנו לגלות האם הבין מהי חלוקת התפקידים בין המחלקות.

בסעיף ג בבגרות איננו מעוניינים לרמוז לתלמידים היכן תוגדר הפעולה / $sumExpressions(...)$ וכן איננו כותבים במפורש את הכותרת שלה. במקרה כזה אנו מוסיפים שלוש נקודות בסוגריים שאחרי שם הפעולה, כדי לפתוח את האפשרות שהפעולה מקבלת פרמטרים.

המושג של 'הגדר בסביבת העבודה' מכוון למעשה למתן הייצוג שבו בחר התלמיד עבור המחלקה. לכן הנוסח הסופי המתקבל כשממירים שאלה זו לנוסחי היחידה החדשים הוא הנוסח המצוי בספר לתלמיד.