

פרק 5

פענוח צפונות ה-main(...)

כל תוכנית ג'אווה כוללת את השיטה הראשית main(...), שממנה מתחיל ביצוע התוכנית. במבט ראשון החתימה של שיטה זו נראית לא ברורה:

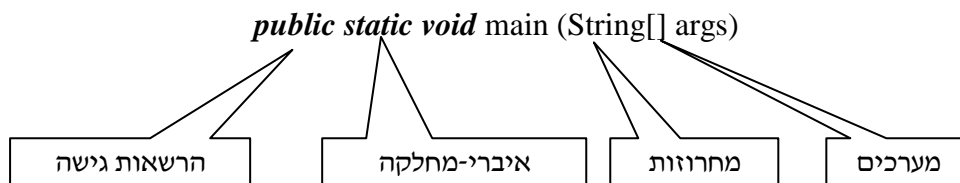
```
public static void main (String[] args)
```

? חלק מן המילים המופיעות בחתימת השיטה ודאי מוכרות לכם. על בסיס מילים אלו, נסו להסיק את מרב הפרטים על השיטה.

בפרק זה נכיר ארבעה כלים שימושיים בשפת ג'אווה, החיוניים להמשך הלימוד:

- מערכים
- מחרוזות
- איברי-מחלקה
- הרשאות גישה

כל הנושאים הללו קשורים לחתימת השיטה main(...), ולכן, בסיום הפרק, נוכל להבין את כל חלקיה של חתימה זו.

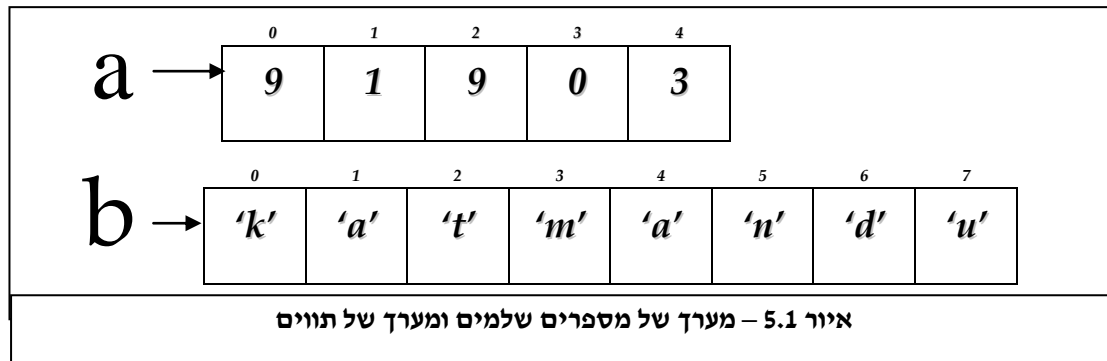


א. מערכים חד-ממדיים

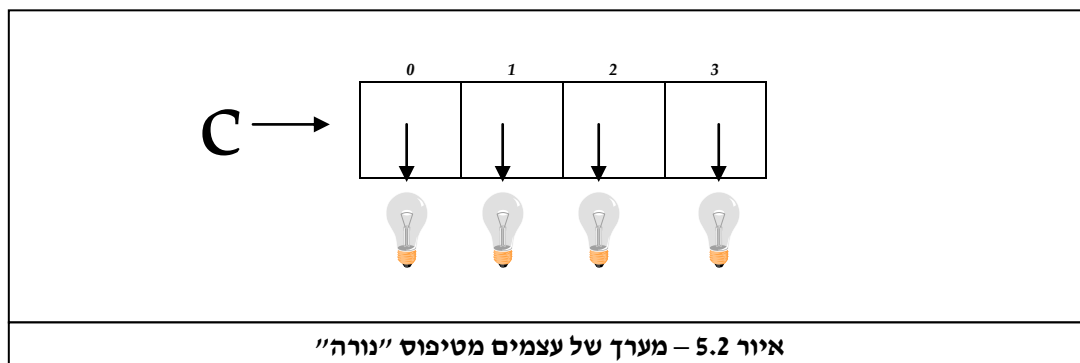
מהו מערך?

מערך (array) הוא טיפוס המייצג אוסף סדור, באורך קבוע, של משתנים מטיפוס זהה. בג'אווה, כל מערך שאנו יוצרים, הוא עצם, ואינו משתנה מטיפוס בסיסי.

לפנינו שני מופעים של מערך: מופע a מציין אוסף סדור של משתנים מטיפוס *int*, ומופע b מציין אוסף סדור של משתנים מטיפוס *char*. כפי שניתן לראות, בתאים של שני המערכים הצבנו ערכים מהטיפוסים המתאימים:



תאי המערך ממוספרים על פי מיקומם. מיקומו של התא הראשון הוא 0, מיקומו של התא השני הוא 1, ומיקומו של התא האחרון הוא $n-1$ (n מציין את מספר התאים במערך). הצגנו שני מערכים של טיפוסים בסיסיים, אולם מערך יכול להכיל גם עצמים. דוגמה לכך היא מערך c, המכיל עצמים מטיפוס "נורה". כל אחד מהתאים במיקום 0-3 מכיל מופע של המחלקה נורה.



length – התכונה האחראית על שמירת אורך המערך

את המערך, כמו כל עצם אחר, מאפיינות תכונות ושיטות. אורך המערך, כלומר מספר תאיו, מיוצג על ידי התכונה *length*. תכונה זו מוגדרת כ-*public*, ולכן נוכל לקבל את ערכה על ידי פנייה אליה באופן הבא:

```
a.length
```

ערכה של *length* נקבע בעת יצירת המערך, ואינו ניתן לשינוי לאחר מכן. אם בשלב מאוחר יותר נרצה להגדיל את מספר התאים במערך, נאלץ ליצור מערך חדש, גדול יותר, ולהעתיק אליו את תוכן המערך הקיים.

טיפוס של מערך

בג'אווה, כמו גם בשפות תכנות אחרות, משמשים הסוגריים המרובעים [] לסימון מערכים. כדי להגדיר טיפוס של מערך נצהיר על טיפוס האיברים שלו, ומימין לשם הטיפוס נוסף סוגריים מרובעים. להלן שלוש דוגמאות לטיפוסים של מערכים:

<u>טיפוס המערך</u>	<u>כיצד נכתוב זאת בג'אווה?</u>	<u>כיצד נקרא זאת?</u>
מערך של מספרים עשרוניים	<code>double[]</code>	<code>double array</code>
מערך של משתנים בוליאניים	<code>boolean[]</code>	<code>boolean array</code>
מערך של עצמים מטיפוס שעות	<code>Clock[]</code>	<code>Clock array</code>

כאשר אנו מצהירים על מערך, אנו מצהירים למעשה על מערך מטיפוס מסוים. אורך המערך נקבע, לעומת זאת, רק בעת יצירתו. טיפוס המערך וגודלו אינם ניתנים לשינוי.

גישה לתאי המערך

כדי לגשת אל המשתנה המוצב בתא במערך, נכתוב את שם המערך ואחריו בסוגריים מרובעים את מיקום התא. לדוגמה, על מנת לגשת אל המשתנה המוצב בתא הראשון במערך בשם `a`, נכתוב:

`a[0]`

לעתים נעדיף לציין את מספרו של התא באמצעות משתנה מטיפוס מספר שלם. לדוגמה, אם נרצה לקבל את הערך שבתא מס' `j` במערך `a` נכתוב `a[j]`.

חריגה מתחומי המערך

אם ננסה לגשת לתא שאינו קיים במערך (למשל: `parkingArea[-2]`), נקבל את "הודעת החריגה" הבאה, המעידה כי ניסינו לגשת למיקום החורג מגבולות המערך:

`ArrayIndexOutOfBoundsException`

? מה יהיה הפלט של קטע הקוד הבא, אם הוא יופעל על המערך `b` שהוצג באיור 5.1:

```
for (int j = 0; j < b.length; j++) {  
    System.out.print (b[j]);  
}
```

? מה נקבל אם נריץ את לולאת ה-`for` האחרונה לאחר שביצענו את השינויים הבאים:

```
b[0] = 'p';  
b[2] = 's';  
b[3] = 'p';  
b[5] = 'r';  
b[6] = 't';
```

? האם שימוש בביטוי `a[a.length]` יגרום לשגיאה? הסבירו.

יצירת מערך של ערכים מטיפוסים בסיסיים

אופן יצירת מערכים דומה מאוד לאופן יצירת עצמים רגילים, אולם ביצירה זו יש כמה קיצורי דרך שמטרתם להקל על המתכנת. נדגים זאת באמצעות יצירת מערך של ערכים מטיפוסים בסיסיים. המערך שניצור ייצג גיליון ציונים ובו עשרה ציונים. בשלב ראשון נצהיר על משתנה בשם `gradeSheet` מטיפוס "מערך של שלמים":

```
int[] gradeSheet;
```

כעת ניצור עצם מטיפוס מערך, ונציב אותו במשתנה `gradeSheet`:

```
gradeSheet = new int[10];
```



כדי להקל על המתכנת, ג'אווה מאפשרת להפעיל את השיטה-הבונה של מערכים באמצעות שימוש בתחביר מיוחד: לאחר המילה השמורה `new` יש לרשום את טיפוס המערך (במקרה זה `int[]`), ולציין את אורך המערך (כאן – 10) בתוך הסוגריים המרובעים. בתגובה לפקודה זו ייבנה עצם מטיפוס מערך של שלמים, שאורכו 10. לאחר מכן תבוצע ההשמה, כלומר המערך החדש יוצב במשתנה מהטיפוס התואם, שעליו הצהרנו קודם (במקרה זה `gradeSheet`). לא הגדרנו מה יהיה ערכם של התאים במערך, ולכן כברירת מחדל יוצב בהם הערך 0 (כפי שניתן לראות באיור 5.3). ניתן להצהיר על מערך ולבנות אותו באותה הפקודה, כפי שעשינו עם עצמים אחרים:

```
int[] gradeSheet = new int[10];
```

יצירת מערך של עצמים

כדי ליצור מערך של עצמים, נשתמש באותו תחביר שהשתמשנו בו כדי ליצור מערך של ערכים מטיפוסים בסיסיים, כפי שפורט בסעיף הקודם:

```
Car[] parkingArea;
```

```
parkingArea = new Car[3];
```

או לחלופין, הצהרה ויצירת עצם באותה השורה:

```
Car[] parkingArea = new Car[3];
```

כאשר יוצרים מערך, רצוי לאתחל את תאיו באופן מפורש ולא להשאיר זאת לשפת ג'אווה. במקרה של מערך של עצמים, כל עוד לא הצבנו בתאי המערך עצמים, המשתנים בתאים יכילו את ערך ברירת המחדל `null`, ונוכל לגשת לתא במערך ולבדוק אם הערך המוצב בו הוא `null`.

את המערך parkingArea, שיצרנו קודם, נוכל לאתחל כך :

```
for (int i = 0; i < parkingArea.length; i++) {  
    parkingArea[i] = new Car();  
}
```

שימו לב, שאמנם ניתן לבדוק אם הערך שמשתנה מפנה אליו הוא *null*, אולם ניסיון לגשת אל איברים של עצם באמצעות הפנייה המכילה *null* יוביל ל"הודעת חריגה" של ג'אווה :
NullPointerException

? כתבו קטע קוד המאתחל את תאי המערך gradeSheet.

? מדוע איננו נדרשים להגדיר את אורך המערך בזמן ההצהרה על טיפוס המערך?

אתחול מערך על ידי שימוש ברשימת אתחול

לעתים, כבר בשלב ההצהרה על המערך, אנו יודעים מה יהיו הערכים שיוצבו בתאי המערך. למשל, אם המערך מייצג את הספרות העשרוניות, ברור לגמרי שהתא הראשון של המערך יכול את הערך 0, התא השני יכול את הערך 1 וכך הלאה. במקרים כאלו נאתחל מערך על ידי ציון רשימת ערכיו במפורש, כך :

```
int[] digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

מספר הערכים ברשימת האתחול קובע את אורך המערך. בדוגמה האחרונה אורכו של המערך המאותחל הוא 10. באגף ימין של שורת הקוד האחרונה בנינו מערך ואתחלנו את תאיו תוך שימוש בתחביר מיוחד, מבלי להשתמש במפורש בפקודה *new*. באותו תחביר ניתן להשתמש גם כדי לאתחל מערך של עצמים שאינם מטיפוס בסיסי.

? בנו את המערך digits ואתחלו אותו מבלי להשתמש ברשימת אתחול.

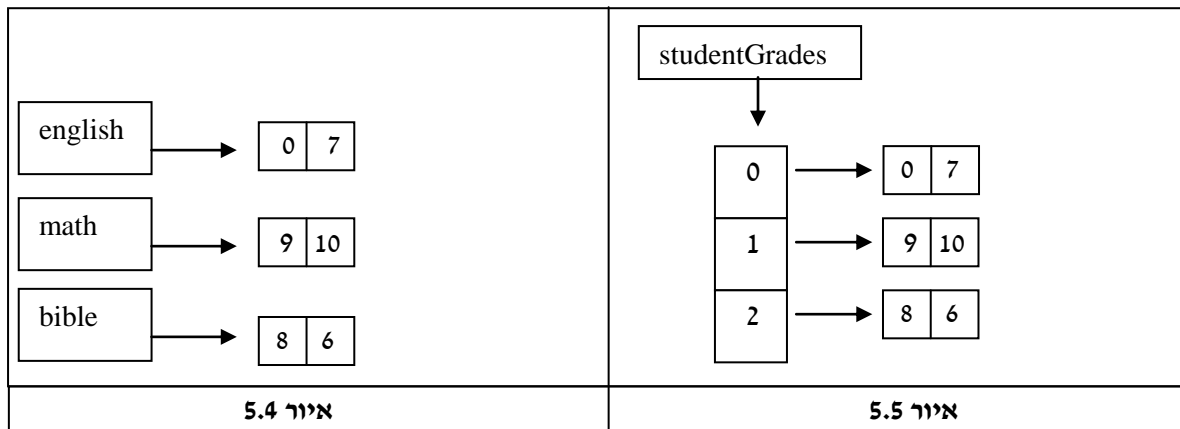
ב. מערכים דו-ממדיים

למדנו כי מערך הוא עצם, וכי תאי המערך יכולים להכיל עצמים. בסעיף זה נעסוק במערכים שהתאים שלהם מכילים עצמים מטיפוס מערך, כלומר נעסוק ב"מערכים של מערכים".

כיצד בנוי "מערך של מערכים"

כדי לאחסן ציונים שתלמיד קיבל עבור עבודותיו בשלושה מקצועות: אנגלית, מתמטיקה ותני"ך, נוכל להגדיר שלושה מערכים מטיפוס *int* (ראו איור 5.4).
אם נרצה לאחסן את הציונים במקצועות השונים במבנה אחד, נוכל להגדיר מערך דו-ממדי דוגמת זה המופיע באיור 5.5.

המערך המופיע באיור 5.5 מורכב משלושה תאים: בתא הראשון – ציוני התלמיד באנגלית, בתא השני – ציוניו במתמטיקה, ובתא האחרון – ציוניו בתנ"ך. זהו למעשה "מערך של מערכים" (בכל תא מוצב מערך) או בקיצור **מערך דו-ממדי (bidimensional array)**.



שימו לב, כי כל אחד מהמערכים english, math ו-bible הוא מערך המכיל שלמים, ולכן הם מטיפוס `int[]`. לעומתם, המערך studentGrades מכיל "מערכים של שלמים", ולכן הוא מטיפוס `int[][]`.

`int[]`
מערך שתאיו מטיפוס `int`

`int[][]`
מערך שתאיו מטיפוס `int[]`

דוגמאות נוספות למערכים דו-ממדיים:

כיצד נכתוב זאת בג'אווה?

טיפוס המערך

`double[][]`

מערך של מערכים של מספרים עשרוניים

`boolean[][]`

מערך של מערכים של משתנים בוליאניים

`Car[][]`

מערך של מערכים של עצמים מטיפוס "מכונית"

כאמור מערך דו-ממדי הוא מערך של מערכים. כדי להקל על הדיון נתייחס למערך דו-ממדי כמערך של שורות.

פנייה לתא של מערך דו-ממדי

כדי לפנות אל תא במערך דו-ממדי, נכתוב את שם המערך, ונוסיף אחריו שני זוגות סוגריים מרובעים: בשמאליים נכתוב את מספר השורה, ובימניים נכתוב את מספר התא בשורה.

[מס' התא בשורה] [מס' השורה] שם המערך

לדוגמה, הפקודה הבאה תדפיס את הערך של התא השני בשורה השלישית (את הערך 6):

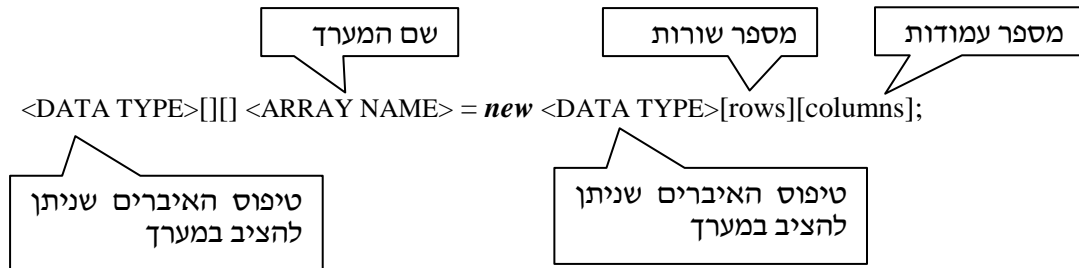
```
System.out.print (studentGrades[2][1]);
```

וכדי להציב בתא זה את ערכו של התא הראשון בשורה השנייה נכתוב:

```
studentGrades[2][1] = studentGrades[1][0];
```

בניית מערכים דו-ממדיים ואתחולם

כדי ליצור מערכים דו-ממדיים נכתוב:



לדוגמה, כדי ליצור את המערך `studentGrades`, מערך דו-ממדי שתאיו מכילים מספרים שלמים (כפי שמופיע באיור 5.5), נכתוב את הפקודה הבאה:

```
int[][] studentGrades = new int[3][2];
```

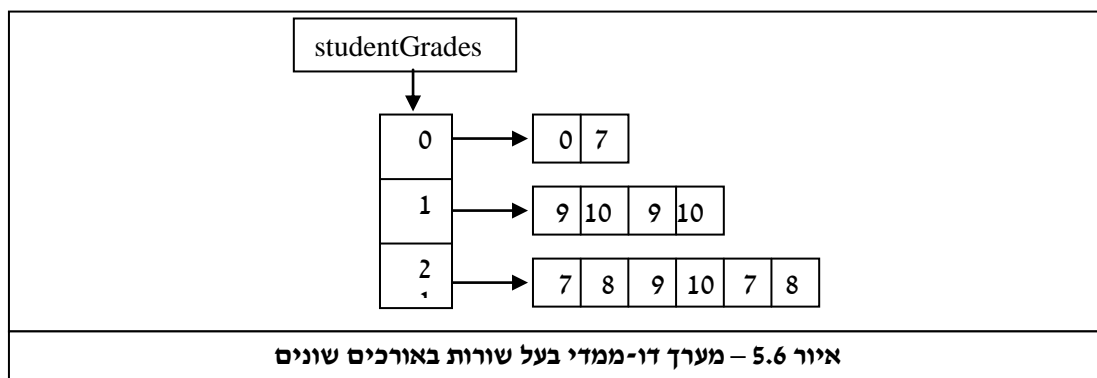
משמאל לסימן השוויון, הצהרנו על המשתנה `studentGrades`, שהוא מטיפוס מערך דו-ממדי של מספרים שלמים. מימין לסימן השוויון יצרנו את המערך הדו-ממדי, תוך ציון טיפוס הנתונים שהוא יכול (`int`) ומספר השורות והעמודות של המערך. נוכל, כמובן, לחלק את הפקודה שמופיעה למעלה לשתי פקודות נפרדות:

```
int[][] studentGrades;
```

```
studentGrades = new int[3][2];
```

בניית מערכים דו-ממדיים שאינם מלבניים

עד כה ראינו כיצד ניתן ליצור מערכים דו-ממדיים מלבניים, כלומר מערכים דו-ממדיים שאורך שורותיהם זהה (ראו איור 5.5). אולם, לעתים נרצה ליצור מערכים דו-ממדיים שאינם מלבניים, דוגמת המערך `studentGrades2` המופיע באיור הבא:



כדי ליצור מערך מלבני בעל שורות באורכים שונים נבצע את הפעולות הבאות :

א. ניצור מערך דו-ממדי, ונקבע את מספר שורותיו בלבד (המערך האנכי באיור 5.6) :

```
int[][] studentGrades2 = new int[3][];
```

בפקודה זו יצרנו למעשה מערך שלו שלושה תאים, שכל אחד מהם יכול להכיל מערך מטיפוס "מערך של שלמים" (בשלב זה כל אחד מהתאים מכיל *null*).

ב. בתאי המערך נציב מערכים חד-ממדיים באורכים הרצויים לנו (המערכים האופקיים באיור 5.6). למשל, כדי להציב בתא השלישי מערך של שלמים באורך של שישה תאים נכתוב :

```
studentGrades2[2] = new int[6];
```

? כתבו קטע קוד היוצר את המערך הדו-ממדי המופיע באיור 5.6.

בניית מערכים דו-ממדיים באמצעות רשימת אתחול

ניתן ליצור מערך דו-ממדי באמצעות רשימת אתחול. את השורות נבנה כפי שלמדנו בעבר, בתוך סוגריים מסולסלים התוחמים רשימת איברים המופרדים זה מזה בפסיקים. לאחר מכן, נפריד בפסיקים את השורות שיצרנו, ו"נעטוף" את הכול בסוגריים מסולסלים נוספים :

```
int[][] studentGrades2 = {{0, 7}, {9, 10, 9, 10}, {7, 8, 9, 10, 7, 8}};
```

? בג'אווה ניתן ליצור מערכים בעלי מספר ממדים גדול כרצונכם.
כיצד תצרו מערך בעל שלושה ממדים?

ג. המחלקה מחרוזת – class String

מחרוזת היא עצם

בשפות תכנות רבות, מחרוזת – סדרה של תווים – היא טיפוס בסיסי. בג'אווה לעומת זאת, מחרוזת היא עצם שנוצר מהמחלקה String, הכלולה בספריית המחלקות של ג'אווה API. דוגמאות למחרוזות בג'אווה: "you are my sunshine", "Yabadabadoo", "123", "a", "" . כל סדרת תווים (גם ריקה!) התחומה בין מירכאות מייצגת מחרוזת.

מחרוזת לעומת טיפוסים בסיסיים

חשוב להבחין בין מחרוזות לבין טיפוסים בסיסיים הדומים להן: 'a' הוא מטיפוס בסיסי *char* אך "a" הוא מחרוזת. 123 הוא מהטיפוס הבסיסי *int* אך "123" הוא מחרוזת. 123.55 הוא מטיפוס בסיסי *double*, ואילו "123.55" – באופן צפוי למדי – הוא מחרוזת.

יצירת מחרוזות

במחלקה String מוגדרות שיטות-בונות רבות. נשתמש בשתיים מהן לבניית מחרוזות:

String()	שיטה-בונה היוצרת מחרוזת ריקה (מחרוזת ללא תווים)
String (String original)	שיטה-בונה היוצרת מחרוזת חדשה שהיא העתק של המחרוזת המועברת

```
String string1 = new String ("This is my String.");
```

```
String string2 = new String();
```

? מה יכילו המשתנים string1 ו-string2 בתום ביצוע שתי השורות האחרונות?

בשל השימוש הנפוץ במחרוזות, ג'אווה מאפשרת ליצור מחרוזת גם ללא אֶזְכוּר מפורש של המילה *new*, על ידי כתיבת רצף התווים הדרוש בין מירכאות, באופן הבא:

```
String string2 = "look: no need to use the word new";
```

האופרטור +

האופרטור הבינרי + יכול לשמש לשתי מטרות:

1. חיבור מספרים:

```
int a = 33;
```

```
int c = a + 1;
```

2. שרשור מחרוזות:

```
String s = "The student's name is Albert" + "Einstein";
```

```
String s2 = s + ".";
```

המשמעות שיקבל האופרטור + תלויה בטיפוסי האופרנדים שעליהם הוא מופעל:

- אם שני האופרנדים הם מספריים – האופרטור **יחבר** ביניהם.
- אם שני האופרנדים הם מחרוזות – האופרטור **ישרשר** אותם.
- אם רק אחד משני האופרנדים הוא מחרוזת, יומר האופרנד השני למחרוזת, והאופרטור ישרשר אותם (משתנה מטיפוס *boolean* יומר למחרוזת "true" או "false" בהתאם לערכו, ולאחר מכן ישורשר).

האופרטור += הופך את האופרנד שמשמאלו לסכום שלו או לשרשר שלו עם האופרנד הימני, בהתאם לאותם הכללים.

סיכמנו את הכללים של האופרטור + בטבלה הבאה:

האופרנד השמאלי		האופרנד הימני	+
מספר	+	מספר	חיבור
מחרוזת		מחרוזת	שרשר
לא מחרוזת		מחרוזת	
מחרוזת		לא מחרוזת	

שיטות המחלקה String

גיאווה מרכזת במחלקה String מגוון רחב של שיטות לטיפול במחרוזות, לדוגמה:

<code>int length()</code>	מחזירה את מספר התווים במחרוזת (כולל רווחים)
<code>char charAt (int index)</code>	מחזירה את התו הנמצא במחרוזת באינדקס הנתון

? מה יהיה ערך המשתנה number בתום ביצוע קטע הקוד הבא:

```
int number = "123123123".length();
number += "".length();
```

שיטות אחרות מאפשרות חיפוש והחלפה של תווים במחרוזת, קבלת קטעים ממחרוזת, המרת אותיות אנגליות קטנות במחרוזת לגדולות ועוד. גיאווה מעמידה לרשותנו מחלקות נוספות לטיפול במחרוזות, אשר מקבלות מחרוזות כקלט, ומבצעות עליהן פעולות שונות:

- StringBuffer – מאפשרת לשנות מחרוזות באופן גמיש יותר.
- StringTokenizer – מאפשרת לפרק מחרוזות לחלקים (parsing) על פי הגדרות חלוקה רצויות (תווים שונים וכד').

פירוט מלא של מחלקות אלה ניתן למצוא ב-Java API.

השיטה toString()

השיטה toString() אינה שיטה של המחלקה String, אלא שיטה המחזירה עצם מטיפוס String. בגיאוה קיימת דרך אחידה לייצוג עצמים על ידי מחרוזות. בכל מחלקה מקובל להגדיר שיטה בשם toString() המחזירה מחרוזות מתאימה שמתארת את העצם. חתימת השיטה היא:

```
public String toString()
```

שיטה זו בונה מחרוזת המייצגת את העצם ומחזירה אותה. השיטה עצמה אינה מדפיסה את המחרוזת.

לדוגמה: נניח שהגדרנו מחלקה בשם A שלה שתי תכונות: x ו-y. השיטה toString() של המחלקה A יכולה להיראות כך:

```
public String toString(){
```

```
    String str = "";
```

```
    str = "A: the value of x - " + x + "the value of y - " + y;
```

```
    return str;
```

```
}
```

השיטה מאתחלת מחרוזת ריקה, בונה אותה בהתאם לייצוג של העצם, ומחזירה אותה למי שזימן השיטה.

עתה, אם ברצוננו להדפיס את המחרוזת שמייצגת את העצם, נוכל לזמן את השיטה, לקבל מחרוזת ולהדפיס אותה.

לדוגמה, נדפיס מחרוזת המייצגת עצם מטיפוס A מתוך השיטה הראשית:

```
public static void main(String[] args){
```

```
    A a = new A();
```

```
    System.out.println (a.toString());
```

```
}
```

לנוחות המתכנת, ניתן לקצר ולכתוב:

```
System.out.println (a);
```

בדרך זו גיאוה מזמנת את השיטה toString() של העצם.

ד. על הפקודה *import* וקצת על חבילות

המחלקות בספריית המחלקות של ג'אווה מאוגדות בחבילות. **חבילה** (*package*) היא אוסף של מחלקות באותו נושא. אם רוצים להשתמש במחלקה מסוימת לצורך התוכנית שלנו, כלומר ליצור עצמים מסוגה, לזמן שיטות שלה וכדומה, אזי יש "לייבא" אותה. "לייבוא" זה נעשה בראש הקובץ המסוים שלנו, שמכיל קוד המשתמש במחלקה המיובאת. איך כל זה מתרחש בפועל? יש לכתוב את המילה השמורה *import* ואחריה את השם המלא של המחלקה (לרבות החבילה שבה היא נמצאת). לדוגמה, המחלקה StringTokenizer שייכת לחבילה java.util. כדי "לייבא" אותה נוסף בראש הקובץ את הפקודה:

```
import java.util.StringTokenizer;
```

אם נרצה להשתמש במחלקות אחדות מהחבילה java.util, נכתוב את הפקודה הבאה אשר תייבא את כל המחלקות הכלולות בחבילה זו:

```
import java.util.*;
```

כאשר אנו מעוניינים להשתמש במספר מחלקות מחבילות שונות, נכתוב כמה פקודות *import* בזו אחר זו. חשוב להקפיד שהפקודות תופענה בתחילת הקובץ, לפני כותרת המחלקה. בפקודות אלו אנו מודיעים למהדר היכן נמצאות המחלקות שבהן אנו משתמשים בתוכנית. שימו לב, הפקודה *import* אינה מוסיפה את קוד המחלקה שבה אנו רוצים להשתמש אל המחלקה שאנו כותבים, אלא רק מציינת לג'אווה היכן נמצא קובץ המחלקה שבה אנו רוצים להשתמש. נשאלת השאלה מדוע לא נדרשנו לייבא לתוכנית גם את המחלקות String ו-StringBuffer כדי להשתמש בהן? והתשובה לכך היא שהמחלקות String ו-StringBuffer שייכות לחבילה java.lang, שאליה משתייכות גם מחלקות אחרות כמו Math ו-System. את החבילה הזו, הכוללת את מחלקות היסוד של שפת ג'אווה, מייבא המהדר באופן אוטומטי, ולכן אין אנו נדרשים לייבא אותה באופן מפורש.

ה. איברי-מחלקה (Class Members)

בסעיף זה נערוך היכרות ראשונה עם **איברי-מחלקה**, שהם איברים השייכים למחלקה ולא למופע מסוים שלה. תחילה נלמד מהן תכונות-מחלקה, ובהמשך נערוך היכרות עם שיטות-מחלקה.

כל התכונות שהכרנו עד כה קרויות תכונות מופע, משום שהן שייכות תמיד לאובייקט מסוים, ומשקפות את מצבו. לעומתן, קיימות תכונות מסוג אחר, אשר אינן שייכות למופע מסוים של המחלקה, אלא למחלקה באופן כללי. ערכי תכונות אלו משקפים את מצבה הנוכחי של המחלקה, ולא את מצבו של מופע זה או אחר שלה. תכונות אלו קרויות תכונות-מחלקה.

דוגמה: מספר סידורי למופעים

נניח שברצונכם לשכלל את אחת המחלקות שכתבתם, כך שכל עצמיה ימוספרו לפי סדר בנייתם, באופן הבא: העצם שיווצר ראשון יקבל את המספר הסידורי 1, העצם שיווצר אחריו יקבל את המספר הסידורי 2, וכך הלאה. סביר להניח, שכדי לעשות זאת תבחרו להוסיף להגדרת המחלקה את שתי התכונות הבאות:

- myIndex – המספר הסידורי של המופע המסוים.
- counter – מספר המופעים שנוצרו עד כה מהמחלקה.

בתחילת ההרצה של התוכנית נציב את הערך 0 בשתי התכונות. בכל פעם שייבנה עצם, בעזרת השיטה-הבונה, נבצע את שתי הפעולות הבאות:

- נקדם ב-1 את counter, משום שמספר המופעים שנוצרו עד כה מהמחלקה גדל ב-1.
- נציב בתכונה myIndex, המציינת את המספר הסידורי של העצם, את הערך הנוכחי של counter, משום שהמספר הסידורי של העצם שווה למספר המופעים שנוצרו עד כה מהמחלקה.

ייתכן שבמבט ראשון שתי התכונות נראות דומות, אולם קיים שוני מהותי ביניהן: התכונה myIndex, כמו כל התכונות שהעסיקו אותנו עד כה, מאפיינת מופע מסוים של המחלקה. לכל אחד מהעצמים במחלקה יש מספר סידורי משלו, כלומר ערך משלו לתכונה myIndex. לעומתה, התכונה counter אמורה לשמור את מספר המופעים שנבנו מהמחלקה עד כה. התכונה אינה מאפיינת עצם מסוים זה או אחר, אלא את המחלקה כולה. כל מופעי המחלקה צריכים להתייחס לאותו counter, ולכן בתכונה counter לא נשמר ערך נפרד בכל מופע, כמו כל התכונות שהכרנו עד כה, אלא נשמר ערך אחד במקום אחד בלבד, במחלקה ולא באובייקטים. לצורך זה ניתן להגדיר בג'אוה תכונות-מחלקה.

תכונות מופע ותכונות-מחלקה

נגדיר אם כן:

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

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

• **תכונת-מחלקה (class variable):**

1. מאפיינת את המחלקה באופן כללי ולא מופע מסוים שלה.
2. בזיכרון קיים עותק יחיד של תכונה זו, ללא קשר למספר המופעים שנוצרו ממנה. לנקודה זו יש שתי השלכות:
 - א. תכונת-מחלקה קיימת עוד לפני שנוצר המופע הראשון.
 - ב. אם מופע מסוים שינה את ערכה של התכונה, ערכה השתנה עבור כל המופעים.

? מדוע איננו יכולים להגדיר את counter כתכונת-מופע?

כפי שראינו, התכונה counter היא דוגמה טובה לתכונת-מחלקה: מספר המופעים שנוצרו עד כה ממחלקה מסוימת הוא אפיון של המחלקה עצמה, ולא של מופע מסוים שלה. שימו לב כי בניגוד לתכונת-מופע, גם בטרם נוצר עצם כלשהו מהמחלקה, תכונת-מחלקה זו קיימת, וניתן לפנות אליה.

איך כותבים את זה בג'אווה?

כדי להגדיר תכונת-מחלקה בג'אווה נקדים להגדרת התכונה את המילה השמורה *static*. אם ננהג אחרת, התכונה המוגדרת תהיה תכונת-מופע ולא תכונת-מחלקה. נציג לדוגמה את קטע הקוד שמוגדרות בו שתי התכונות של המחלקה הממספרת את מופעיה:

```
/** This class counts the number of its instances */
```

```
public class InstanceCounter{  
    public static int counter = 0; // counts the instances created so far (class variable)  
    private int myIndex = 0; // the index of this instance (instance variable)  
}
```

הקדמנו להגדרת התכונה counter את המילה השמורה *static*, ובכך הגדרנו אותה כתכונת-מחלקה, או "תכונה סטטית". משמעות הדבר היא, שקיים עותק אחד בלבד של תכונה זו במחלקה ולא עותק בכל אובייקט. לעומתה, myIndex היא תכונת-מופע, ואינה תכונת-מחלקה, ולכן לכל אחד מהעצמים של המחלקה יהיה העתק נפרד שלה.

עוד בטרם נוצר המופע הראשון כבר קיימת תכונת המחלקה counter, שאותחלה על ידי ג'אווה לערך ברירת המחדל המתאים לטיפוס המשתנה.

עם היווצרות המופע הראשון של המחלקה ערך counter עולה ב-1. השיטה-הבונה מעלה את ערך counter ב-1, ומציבה את ערך תכונת המחלקה counter בתכונה myIndex של המופע החדש שהיא יוצרת.

פנייה לתכונות-מחלקה בג'אווה

פנייה לתכונת-מחלקה נעשית באחד משני אופנים:

הדרך העדיפה היא ציון שם המחלקה ואחריה ציון שם התכונה, והפרדתם בעזרת סימון-הנקודה. נציג לדוגמה שיטה במחלקה InstanceCounter, אשר מדפיסה את מספרו הסידורי של המופע, ואחריו את מספר העצמים שנוצרו עד כה מהמחלקה.

```
public void printNumbers(){
    String s = "So far " + InstanceCounter.counter + "objects were created. " +
        " My number is " + this.myIndex;
    System.out.println (s);
}
```

דרך אחרת היא פנייה למופע. בג'אווה כל מופע "מכיר" את המחלקה שלו, ויכול להתייחס לתכונותיה. לכן, ניתן היה להחליף את הפנייה InstanceCounter.counter המופיעה בקטע הקוד האחרון, בפנייה מהצורה הבאה:

```
this.counter
```

גם עצמים ממחלקות אחרות יכולים לפנות לתכונת-מחלקה בשני האופנים שראינו, בתנאי, כמובן, שהרשאת הגישה של התכונה היא public. דוגמה לכך מופיעה בקטע הקוד הבא, הלקוח מתוך שיטה של מחלקה אחרת:

```
InstanceCounter ic = new InstanceCounter();
int j = ic.counter;
```

לאחר שנוצר המופע ic, מתייחס ic.counter לערך של התכונה counter, שהיא תכונת-מחלקה של מחלקת InstanceCounter.

? במהלך ריצת תוכנית נוצרו חמישה מופעים מטיפוס InstanceCounter. לאחר מכן הפעילו את השיטה printNumbers() על המופע שנוצר רביעי. מה תדפיס השיטה?

לסיכום נחזור ונזכיר כי כל אחד ממופעי המחלקה "מסתכל" על אותו עותק יחיד של התכונה counter המשותף לכולם. אם אחד המופעים, או גורם חיצוני, ישנה את ערך התכונה counter, אזי ערך התכונה ישתנה בכל המופעים. לעומת זאת, לכל אחד מהמופעים יש עותק משלו של התכונה myIndex. בעותק זה מוצב ערכו של counter כפי שהיה בעת היווצרות המופע. הצגנו שתי דרכים לפנייה אל תכונת-מחלקה. ככלל, נעדיף לפנות לתכונה סטטית דרך המחלקה, משום שבכך אנו מדגישים את היותה תכונת-מחלקה ולא תכונת-מופע.

תכונות קבועות ותכונות-מחלקה

פעמים רבות נעדיף להגדיר תכונה קבועה, שהוגדרה על ידי המילה השמורה *final* כתכונת-מחלקה ולא כתכונה של מופע. הסיבה לכך פשוטה: ערכו של קבוע נקבע פעם אחת. לכן ניתן להסתפק בעותק יחיד של הקבוע לכל המופעים, ולא לשמור העתקים זהים של התכונה בכל אחד מהמופעים. כדי לקבל עותק אחד ויחיד של התכונה, נצהיר עליה כעל תכונת-מחלקה באמצעות המילה השמורה *static*.

למשל, אם נרצה להגדיר את הקבוע הפיזיקלי המגדיר את כוח המשיכה של כדור הארץ (*gravity*), נכתוב:

```
public static final double GRAVITY = 9.81;
```

הצהרה זו קובעת שזהו קבוע (*final*), פומבי (*public*), של המחלקה (*static*), והוא מטיפוס הנתונים הבסיסי *double*. הצירוף של *final* ו-*static* הוא צירוף נפוץ מאוד בג'אווה.

שיטות-מופע ושיטות-מחלקה

כל השיטות שעסקנו בהן עד כה היו שיטות-מופע (*instance methods*), כלומר פעולות שמבצע מופע מסוים של המחלקה. שיטות-מחלקה (*class methods*) הן פעולות שהמחלקה מבצעת, ולא מופע מסוים שלה. שיטות כאלו ניתן לזמן גם בלי לבנות מופעים של המחלקה. בדרך כלל ניתקל בשיטות-מחלקה בתוך "מחלקות שירות" – המציעות אוסף של שיטות-מחלקה פומביות בתחומים מוגדרים. דוגמה למחלקת שירות כזו היא המחלקה *Math*.

מחלקות שירות – המחלקה *Math*

בתוכניות רבות קיים צורך לבצע חישובים מתמטיים שונים. במקרים כאלו עומדת לרשות המתכנת המחלקה *Math*, שהיא מחלקת שירות של ג'אווה, המצויה ב-Java API. מחלקה זו היא בעצם "מחשבון כיס" משוכלל מאוד: בין שיטותיה השונות – כולן שיטות-מחלקה – כלולות פעולות מתמטיות מגוונות המבוצעות על ערכי הקלט, כגון: העלאה בחזקה, חישובים לוגריתמיים, חישובי שורשים, פונקציות טריגונומטריות ועוד.

? התבוננו ב-Java API של המחלקה *Math*, ורשמו חמש שיטות-מחלקה שאתם מזהים בה.

איך כותבים את זה בג'אווה?

בדומה לתכונות-מחלקה, גם שיטות-מחלקה מוגדרות בג'אווה באמצעות הוספת המילה השמורה *static* לחתימת השיטה, ולכן הן מכונות "שיטות סטטיות". אם לא נוהגים כך, השיטה המוגדרת תהיה שיטת-מופע ולא שיטת-מחלקה. הקוד המלא של מחלקת InstanceCounter, ייראה אם כן כך:

```
/** This class counts the number of its instances */  
public class InstanceCounter {  
    public static int counter = 0;    //counts the instances created so far (class variable)  
    private int myIndex = 0;        //the index of this instance (instance variable)  
  
    /** Creates a new object */  
    public InstanceCounter(){  
        InstanceCounter.counter++;    //increments the class variable  
        this.myIndex = counter; //assigns the current value of the class variable  
                                    //to the instance variable of this instance  
    }  
    /** Returns the number of instances created so far (a class method) */  
    public static int getCounter(){  
        return InstanceCounter.counter;  
    }  
    /** Returns the index of this instance */  
    public int getIndex(){  
        return this.myIndex;  
    }  
}
```

כדי לקבל את מספר העצמים שנוצרו עד כה נפעיל את שיטת המחלקה `getCounter()`:

```
InstanceCounter.getCounter()
```

איננו זקוקים לעצם מטיפוס המחלקה כדי להפעיל שיטה זו, אם כי אנו רשאים להפעילה על ידי פנייה לעצם.

ניתן היה לממש שיטה זו כשיטת-מופע (על ידי השמטת המילה *static*), אולם לא מומלץ לעשות כן. הערך המוחזר אינו תלוי במופע מסוים, ויתירה מזאת, אנו רוצים להיות מסוגלים להפעיל את השיטה הסטטית עוד לפני שנוצר מופע כלשהו של המחלקה. לעומת זאת, כדי להפעיל את השיטה `getIndex()` נזדקק לעצם מטיפוס המחלקה, והשיטה תחזיר את ערך התכונה המתאימה של העצם המסוים שעליו הופעלה.

שיטות-מחלקה אינן יכולות לגשת לאיברי מופע

שימו לב, במימוש של שיטות-מחלקה אין אפשרות לגשת לאיברי מופע, וניסיון לעשות כן לא יעבור את שלב ההידור. זאת מכיוון ששיטות-מחלקה אינן מופעלות על עצם מסוים, אלא על המחלקה, וכפי שראינו, ניתן לזמן אותן גם ללא בנייה של עצם מסוים. לפיכך, אם במימוש שיטת-מחלקה נפנה לאיבר שאינו איבר-מחלקה (איבר שאינו מוגדר כסטטי), ייתכן שהוא עדיין אינו קיים, ולכן גיאווה תודיע לנו על שגיאת הידור. לסיכום, במימוש של שיטת-מחלקה נוכל לפנות אך ורק לאיברים שהוגדרו כסטטיים.

פנייה לשיטות-מחלקה בג'אווה

כדי לפנות לשיטות-מחלקה, בדומה לדרך הפנייה לתכונות-מחלקה, נציין את שם המחלקה ואחריה את שם השיטה, ונפריד את שני השמות בסימון-הנקודה.

```
System.out.println (InstanceCounter.getCounter());
```

גם במקרה של שיטת-מחלקה ניתן לפנות לשיטה על ידי פנייה למופע של המחלקה :

```
InstanceCounter ic = new InstanceCounter();
```

```
int j = ic.getCounter();
```

הואיל ושיטת המחלקה משויכת למחלקה ולא לעצם מסוים, ניתן ורצוי לפנות לשיטה ישירות דרך המחלקה, ולא דרך מופע מסוים שלה.

שימו לב : ישנן שתי דרכים לקרוא לשיטות מתוך השיטה הראשית main(...) :

- דרך אחת : יצירת מופע של המחלקה, וקריאה לשיטה על ידי פנייה למופע.

- דרך שנייה : קריאה לשיטה סטטית.

לא ניתן לקרוא לשיטה שאינה סטטית בלי ליצור קודם לכן מופע, מפני שהשיטה main(...) עצמה היא שיטה סטטית (זוכרים?) **public static void main(...)**.

שיטות סטטיות מוכרות לכם כבר ממחלקות הקלט והפלט הגרפיות :

InputRequestor, OutputWindow

כל השיטות של המחלקות האלו הן סטטיות, למשל :

```
InputRequetor.requestInt ("Enter an integer: ");
```

1. הרשאות גישה (Access Specifiers)

אנקפסולציה ותכנות מונחה עצמים

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

הרשאות הגישה לאיברים

כאשר אנו מגדירים איבר, עלינו להחליט אילו מחלקות תוכלנה לגשת אליו. הרשאת הגישה נקבעת על פי מגדירי גישה המופיעים במחלקה לפני הגדרת האיבר. בג'אווה קיימות כמה רמות של הרשאות גישה:

1. **פומבי (public)** – אם לפני הגדרת האיבר מופיע מגדיר הגישה *public*, עצמים מכל המחלקות בג'אווה יכולים לגשת אליו.

2. **פרטי (private)** – אם לפני הגדרת האיבר מופיע מגדיר הגישה *private*, ניתן לגשת לאיבר רק מתוך קוד המחלקה שבה הוא מוגדר. במילים אחרות, רק המתכנת של אותה המחלקה יכול, במהלך כתיבת הקוד, להשתמש באיברים הפרטיים.

קיימת הרשאת גישה מרמה שלישית, *protected*, הקשורה למנגנון הירושה, ובה לא נעסוק כאן, אלא בפרק הבא שעניינו ירושה. מהאמור לעיל נובע שניתן לגשת לאיבר של עצם בשני מקרים:

- אם האיבר מוגדר כפומבי.
- אם האיבר מוגדר כפרטי והגישה אליו היא מתוך הקוד של המחלקה שהוא מוגדר בה (האיבר הוא תכונה של המחלקה או שיטה שלה). דוגמה מוכרת לכך היא אתחול תכונות פרטיות בתוך השיטה-הבונה.

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

אליס ובוב

בקטע הבא נעסוק בשתי מחלקות: Alice ו-Bob. המחלקה Alice, המופיעה בצד שמאל, עברה הידור. הכותב של המחלקה Bob לעומת זאת, לא הבין די הצורך את מנגנון הרשאות הגישה, ולכן הוא לא הצליח להדר את הקוד. קוד המחלקה Bob מופיע בצד ימין.

<pre>public class Alice { public int a1; private int a2; public Alice(){ this.a1 = 1; this.a2 = 2; } public void increaseBoth(){ this.a1++; this.a2++; } private int findA2 (Alice a){ return a.a2; } }</pre>	<pre>public class Bob { private Alice ally; public Bob(){ this.ally = new Alice(); } public void changeAlice(){ System.out.println ((this.ally).a1); (this.ally).increaseBoth(); System.out.println ((this.ally).a2); (this.ally).findA2 (this.ally); } }</pre> <div style="margin-top: 10px;"> <p>חוקי השיטה-הבונה של Alice היא פומבית</p> <p>חוקי התכונה a1 של Alice היא פומבית</p> <p>חוקי הפעלת שיטה פומבית של Alice</p> <p>לא חוקי: פנייה לאיברים פרטיים של Alice</p> </div>
<p>איור 5.7 – במחלקה Bob נעשית פנייה לאיברים פרטיים של המחלקה Alice לכן היא לא תעבור הידור</p>	

במחלקה Bob מוגדרת התכונה ally, שהיא מטיפוס המחלקה Alice. כותב המחלקה Bob מאתחל את ally בשיטה-הבונה (על-ידי קריאה לשיטה-הבונה של המחלקה Alice), ובשיטה changeAlice() הוא מנסה לגשת לתכונות ולשיטות של העצם ally.

בשורה הראשונה של שיטה זו יש ניסיון לגשת לתכונה פומבית של ally, ובשורה השנייה יש ניסיון להפעיל שיטה פומבית שלה. כל עוד כותב המחלקה Bob פונה לאיברים המוגדרים כפומביים במחלקה Alice, הוא מורשה לעשות כן, משום שלאיברים פומביים ניתן לגשת גם מחוץ לקוד המחלקה שבה הם מוגדרים. לעומת זאת, בשורה השלישית של השיטה מנסה המתכנת לפנות לתכונה המוגדרת במחלקה Alice כפרטית, ובשורה הרביעית הוא מנסה להפעיל שיטה המוגדרת במחלקה Alice כפרטית. גישה לאיברים פרטיים המוגדרים במחלקה אחרת אינה אפשרית. לכן, המחלקה Bob לא תעבור את שלב ההידור. אל האיברים המוגדרים כפרטיים במחלקה Alice ניתן לפנות רק בתוך קוד המחלקה עצמה.

שימו לב, השיטה findA2 (Alice a), המופיעה במחלקה Alice, מקבלת עצם מטיפוס המחלקה Alice, ובגוף השיטה מתבצעת פנייה לאיבר פרטי של עצם זה – a2. פנייה כזו מותרת, משום שהיא מתבצעת באותה מחלקה שבה מוגדר האיבר הפרטי a2.

הפרדה בין ממשק למימוש

באמצעות מגדירי גישה יכול כותב המחלקה להפריד בצורה קלה בין ממשק המחלקה לבין המימוש שלה. האיברים הפומביים, המוגדרים באמצעות מגדיר הגישה *public*, מרכיבים את ממשק המחלקה. עצמים ממחלקות אחרות יכולים לעבוד ולתקשר עם עצם ממחלקה אחרת רק דרך ממשק זה, על ידי זימון שיטות פומביות או על ידי גישה לתכונות פומביות. לעומתם, האיברים הפרטיים, המוגדרים באמצעות מגדיר הגישה *private*, משמשים רק למימוש המחלקה. ניתן לזמן שיטות פרטיות או לגשת לתכונות פרטיות רק מתוך קוד המחלקה, כלומר אך ורק בעת כתיבת המחלקה.

להפרדה הברורה בין ממשק למימוש יתרון בולט: עצמים של מחלקות אחרות אינם יכולים להשתמש באיברים הפרטיים, ובדרך כלל גם אינם יודעים עליהם. כל עוד הממשק של העצם נשאר זהה, אנו יכולים לשנות ולשפר את המימוש שלו מעת לעת, בלי לפגוע בתוכניות שכבר משתמשות בו.

גישה מבוקרת לתכונות

כיוון שאיננו יכולים לגשת לתכונות פרטיות, נהוג לממש שיטות `getX()` ו-`setX(...)` (בהנחה ש-X הוא שם של תכונה), המאפשרות זאת. במה שונה גישה דרך השיטות הללו מהגישה הישירה לתכונות?

זו גישה מבוקרת. נממש את השיטות האלו רק בעבור התכונות שאנחנו רוצים לאפשר גישה אליהן מבחוץ: עבור קריאה נממש את שיטת `getX()`, ועבור שינוי נממש את השיטה `setX(...)`. בדוגמה של המחלקה איגואנה, כאשר לא רצינו לאפשר שינוי של תאריך יום ההולדת, לא הגדרנו שיטה בשם `setBirthday()`. מאותה סיבה, כאשר מימשנו את השיטה `getBirthday()` החזרנו רק את העותק של התכונה ולא את התכונה ממש. אילו התכונה הייתה פומבית, לא היינו יכולים לספק הגנה כזו.

שביל הזהב – מי פומבי ומי פרטי

לאחר שהבנו מהן הרשאות גישה, נדון בשיקולים המנחים את השימוש בהן. מחלקה שכל איבריה יוגדרו פרטיים תהיה בטוחה במאת האחוזים – איש לא יוכל לפגוע בה. אולם, היא גם תהיה חסרת תועלת לאחרים – הממשק שלה יהיה ריק, ואיש לא יהיה מסוגל להשתמש בה. לעומתה, מחלקה שכל איבריה יוגדרו כפומביים תאפשר לכל החפץ בכך להשתמש בה, אולם היא גם תהיה פגיעה מאוד – כל איבריה יהיו חלק מהממשק שלה, וכולם יורשו לגשת לאיבריה ולהשפיע על ערכי תכונותיה. נשאלת השאלה, היכן עובר הקו המפריד בין ממשק למימוש? מתי נגדיר איבר כפומבי ומתי נגדירו כפרטי? כדי להחליט האם איבר יוגדר פומבי או פרטי נפעל לפי "כללי האצבע" הבאים:

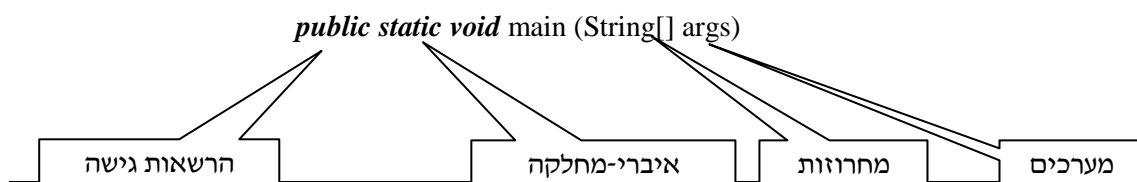
- **שיטות שירות תוגדרנה פומביות** – מחלקות אחרות משתמשות במחלקה על ידי זימון שיטותיה. שיטות אלו, שדרכן המחלקה מספקת את שירותיה, יוגדרו כפומביות, כדי לאפשר למחלקות אחרות לזמן אותן.

- **שיטות עזר תוגדרנה פרטיות** – מלבד שיטות השירות, אלו שדרכן המחלקה מספקת את שירותיה, נמצא בדרך כלל גם שיטות נוספות, המסייעות לכותב המחלקה לממש את שירותיה. שיטות עזר אלו יוגדרו כפרטיות משום שנועדו לכותב המחלקה בלבד.
- **על פי רוב תכונות תוגדרנה פרטיות** – כותב המחלקה צריך למנוע מראש כל אפשרות של שינוי לא רצוי של ערכי התכונות, בין במזיד ובין בשגגה. אחת ממשימותיו החשובות של כל כותב מחלקה היא לשמור על עקביות התכונות, כלומר להיות בטוח כל העת שהערכים המוצבים בתכונות הם הערכים הנכונים והרצויים מבחינתו. כדי להבטיח זאת, כותב המחלקה יגדיר את כל התכונות כפרטיות. אם בכל זאת יהיה מעוניין לאפשר למחלקות אחרות לטפל בתכונות המחלקה שלו, יאפשר להן לעשות כן אך ורק בתיווכו ובידיעתו, כלומר באמצעות שיטות שירות פומביות שיגדיר לשם כך. כאשר הגישה לתכונות היא רק דרך שיטות הנכתבות על ידי מתכנת המחלקה, ניתן להבטיח את עקביות הערכים המוצבים בהן.
- **על פי רוב, תכונות קבועות תוגדרנה פומביות** – ערכן של תכונות קבועות נקבע פעם אחת בלבד בעת כתיבת המחלקה, ולא ניתן לשנות אותן לאחר מכן. מסיבה זו, אין מניעה להגדיר תכונות קבועות כפומביות, משום שממילא כותבי מחלקות אחרות לא יוכלו לשנות את ערכן.

ז. סיכום

בתחילת הפרק הבטחנו כי בסיומו נוכל להבין את חתימת השיטה `main(...)`. הגיעה העת לפרוע את החוב.

? התבוננו בחתימת השיטה `main(...)`, ונסו להסיק את מרב הפרטים על השיטה.



הידע שרכשנו בפרק זה מאפשר לנו להבין על נקלה את כל מרכיבי ה-`main(...)`:

- השיטה `main(...)` היא פומבית (**public**), כלומר ניתן לפנות אליה או לקרוא לה גם מתוך קוד של מחלקה אחרת. זאת בניגוד לשיטות פרטיות (**private**), אליהן ניתן לפנות רק בתוך הקוד של אותה המחלקה.
- בנוסף, השיטה `main(...)` מוגדרת סטטית, כלומר היא שיטת-מחלקה ואינה שיטת-מופע. משמעות הדבר היא שהשיטה אינה מקושרת באופן ספציפי למופע של המחלקה שבה היא שוכנת, אלא למחלקה עצמה. ניתן להפעיל את השיטה גם ללא יצירת מופעים של המחלקה (חשבו, האם נדרשתם לבנות מופע של המחלקה שהכילה `main(...)` כדי להפעיל את השיטה?).

- ערך ההחזרה של `main(...)` הוא `void`, כלומר השיטה מחזירה את הערך הריק (ואת זה ידענו עוד לפני קריאת הפרק).
- השיטה `main(...)` מקבלת כארגומט מערך של מחרוזות. הן מערכים והן מחרוזות בג'אווה הם עצמים לכל דבר, אם כי שניהם "מיוחסים" משהו: בשל השימוש הנפוץ בהם הונהגו לגביהם מספר "קיצורי דרך" תחביריים הייחודיים להם. מערך המחרוזות `args`, משמש לקליטת נתונים מהמשתמש שמריץ את `main(...)`, אולם לא נעסוק בכך כעת.

הנושאים שלמדנו בפרק זה: הרשאות גישה, איברי-מחלקה, מערכים ומחרוזות, מהווים נדבכים נוספים בהכרת התחביר של שפת ג'אווה. הכרתם תאפשר לנו להמשיך ביתר קלות לנדבכים מתקדמים בתכנות מונחה עצמים, שהראשון שבהם, ירושה, יידון בפרק הבא.

מושגים

instance members	איברי מופע
class members	איברי-מחלקה
access specifiers	הרשאות גישה
<code>main(...)</code>	השיטה הראשית
package	חבילה
String	מחרוזת
array	מערך

פרק 5 דף עבודה מס' 1

מערכים ומחרוזות

מטרות

1. שימוש במערכים ומחרוזות.
2. שימוש ב-API של המחלקה String.

מבוא

בתרגיל זה עליכם להשתמש ב-API של מחלקה סטנדרטית של Java. כדי ללמוד איך להשתמש ב-API תוכלו להיעזר במדריך הטכני בסעיף 4. דפי Java API מתעדים מחלקות באופן מוסכם. ניתן ליצור את הדפים בקלות גם למחלקות שאתם כותבים בעזרת מנגנון javadoc (נלמד להשתמש בו בדף עבודה מס' 7).

מה עליכם לעשות?

כדי לבצע חלק מהמטלות הבאות עליכם לעיין ב-API של String ולמצוא שם את השיטות הנחוצות לכם.

1. עליכם ליצור מערך ובו חמש מחרוזות, כל מחרוזת היא שם של אדם (לדוגמה "Nir Cohen").
2. עליכם לכתוב תוכנית המדפיסה את כל השמות המסתיימים באות 'n'.
3. כל האנשים ברשימה החליטו על הוספת שם משפחה שני "Israeli". הוסיפו אותו לכל אחד מהשמות במערך, והדפיסו את המערך.
4. הפכו את כל האותיות במחרוזות שבמערך לאותיות גדולות (לדוגמה, במקום "Nir" כתבו "NIR"), והדפיסו את המערך.

מהצחה!

פרק 5 דף עבודה מס' 2

כיתה מוזיקלית

מטרות

תרגול שימוש בתכונה סטטית.

מה עליכם לעשות?

לפניכם תוכנית המייצגת כיתה שתלמידיה אוהבים מוזיקה. לתלמידים יש ספרייה מוזיקלית משותפת. הם אוספים דיסקים ומשמיעים אותם בהפסקות. בתחילת השנה, כל תלמיד מתחייב להביא כמות מסוימת של דיסקים שונים (INITIAL_AMOUNTS_OF_DISCS), מכאן ואילך הוא יכול להוסיף דיסקים אם רצונו בכך (בסוף השנה הילד שהביא הכי הרבה יקבל פרס מיוחד מבני כיתהו).

רן, חובב המוזיקה והמחשבים, כתב מחלקה המייצגת תלמיד בכיתה, כדי שיוכל לערוך מעקב אחרי כמות הדיסקים שמביאים התלמידים. לפניכם מחלקה שמייצגת תלמיד בכיתה זו:

```
public class ClassMember{
    private String name;
    private int myAmount;
    private static int classDiscBox = 0;
    public static final int INITIAL_AMOUNTS_OF_DISCS = 3;

    public ClassMember (String name){
        this.name = name;
        this.myAmount += INITIAL_AMOUNTS_OF_DISCS;
        ClassMember.classDiscBox += INITIAL_AMOUNTS_OF_DISCS;
    }
    public static int getClassDiscBox(){
        return ClassMember.classDiscBox;
    }
    public int getStudentAmount(){
        return this.myAmount;
    }
    public String getName(){
        return this.name;
    }
    public void enterDiscs (int amount){
```

```

        ClassMember.classDiscBox += amount;
        this.myAmount += amount;
    }
}

```

1. במחלקה אחרת הוגדרה השיטה הראשית. מה תהיה תוצאת ההדפסה?

```

public class Test {
    public static void main (String[] args){
        ClassMember[] members = new ClassMember[3];
        members[0] = new ClassMember ("Moshe");
        members[1] = new ClassMember ("Dvir");
        members[2] = new ClassMember ("Michal");

        for (int i = 0; i < members.length ; i++){
            members[i].enterDiscs (i);
        }
        System.out.println (members[2].getStudentAmount());
        System.out.println (members[2].getClassDiscBox());
    }
}

```

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

בהצלחה!

פרק 5 דף עבודה מס' 3

מערכים – מחלקת שירות

מטרות

תרגול השימוש במערכים המכילים טיפוסים בסיסיים ובמערכים המכילים מחרוזות.

מה עליכם לעשות?

עליכם לכתוב שתי מחלקות:

- `ArrayUtil` – מחלקת שירות הכוללת שיטות סטטיות שימושיות שניתן להפעיל על מערכים.
- `TestArrayUtil` – מחלקה שכוללת את השיטה `main(...)` ושנועדה לבדוק את שיטות המחלקה `ArrayUtil`.

המחלקה `ArrayUtil`

מחלקה זו כוללת מגוון שיטות שימושיות לעבודה עם מערכים מסוגים שונים. במהלך התרגילים הבאים נשתמש ברוב השיטות. כל השיטות במחלקה הן שיטות סטטיות. פירוש הדבר הוא שכדי להשתמש בשיטות המחלקה אין צורך לבנות אובייקט מסוג `ArrayUtil`. אם אינכם זוכרים מהי משמעות המילה השמורה `static`, מהי שיטה סטטית או כיצד משתמשים בה, קראו שוב את הסעיפים המתאימים בפרק. ממשק המחלקה שעליכם לממש:

תיאור השיטה	חתימת השיטה
מדפיסה את תוכן המערך בשורה אחת, עם רווח אחד בין איבר לאיבר (מהאינדקס הקטן אל הגדול)	<code>static void printArray (int[] array)</code>
מחזירה את סכום המספרים במערך	<code>static int getSum (int[] array)</code>
מחזירה את המחרוזת הארוכה ביותר במערך (אם יש כמה מחרוזות באורך זהה, היא מחזירה את הראשונה מביניהן)	<code>static String getLongest (String[] arrayOfStrings)</code>
מדפיסה את תוכן המערך שורה אחרי שורה	<code>static void printArray (int[][] array)</code>
מחזירה את סכום המספרים באלכסון המתחיל בפינה השמאלית-העליונה	<code>static int leftDiagonalSum (int[][] array)</code>
מחזירה את סכום המספרים באלכסון המתחיל בפינה הימנית-העליונה	<code>static int rightDiagonalSum (int[][] array)</code>

שימו לב לכך שבממשק יש שימוש בשלושה סוגי מערכים: `int[]`, `int[][]`, `String[]`.

מומלץ מאוד לבדוק את המחלקה ArrayUtil בשלבים. כלומר, בכל פעם שתסיימו לממש שיטה, בדקו את תקינותה על ידי הפעלתה מתוך המחלקה TestArrayUtil (ראו בסעיף הבא). כדי לבדוק כל שיטה בנפרד מומלץ להתחיל במימוש שיטות ההדפסה של המערכים. בכל ההדפסות הנדרשות בשיטות המפורטות לעיל, יש להשתמש בהדפסה רגילה אל המסך, ולא להיעזר בחלונות פלט גרפיים. במימוש כל השיטות המפורטות לעיל, אתם רשאים להניח שהמערכים הדו-ממדיים הם מערכים ריבועיים.

המחלקה TestArrayUtil

כדי לבדוק את תקינות השיטות שמימשתם, כתבו מחלקה בשם TestArrayUtil. בשיטה main(...) של מחלקה זו, בנו אובייקטים מסוג מערכים והפעילו עליהם את השיטות הסטטיות שכתבתם. פלט התוכנית צריך לכלול הפעלה של כל השיטות שכתבתם.

מהצחה!

פרק 5 דף עבודה מס' 4

משחקי מילים – שימוש ב-String API

מטרות

ללמוד להשתמש במחלקה String הכלולה בספריית המחלקות הסטנדרטית של ג'אווה.

רקע

פלינדרום (palindrome) הוא מילה או משפט הנקראים באותו אופן משמאל-לימין ומימין-לשמאל. לדוגמה, המילה pop היא פלינדרום וכך גם המשפטים הבאים:

Rats live on no evil star

Able was I ere I saw elba

שימו לב, פלינדרום אינו רגיש להבדלי אותיות קטנות או גדולות (case sensitive), לדוגמה המחרוזת aaBbAa היא פלינדרום.

מה עליכם לעשות?

עליכם לכתוב תוכנית המקבלת מחרוזת ומודיעה אם היא פלינדרום או לא.

היעזרו במחלקות הגרפיות InputRequestor ו-OutputWindow.

עליכם לכתוב שתי מחלקות:

- CheckPalindrome – כוללת מספר שיטות המאפשרות לבדוק אם מחרוזת היא פלינדרום.
- PalindromeGame – מקבלת מהמשתמש מחרוזת ומודיעה האם המחרוזת היא פלינדרום.

המחלקה CheckPalindrome

עליכם לממש את הממשק הבא:

חתימת השיטה	תיאור השיטה
<code>static String backwardString (String str)</code>	מחזירה מחרוזת הפוכה מזו שהועברה אליה
<code>static boolean isPalindrome (String strToCheck)</code>	בודקת אם מחרוזת היא פלינדרום*

*תזכורת: פלינדרום אינו רגיש להבדלי אותיות קטנות וגדולות.

כדי לממש את המחלקה השתמשו בשיטות המופיעות בממשק המחלקה String. עיינו בממשק המחלקה. השיטות המופיעות בממשק מסודרות לפי סדר אלפביתי. אם תקראו את חתימת

השיטה ואת ההסבר המופיע לצדה, תוכלו לדעת אם היא מתאימה לצורכיכם וכיצד ניתן להשתמש בה.

הערה: בשביל לממש את המחלקה אין צורך להשתמש ביותר משלוש שיטות של המחלקה String.

המחלקה PalindromeGame

מחלקה זו תציג בפני המשתמש חלון קלט ובו ההודעה הבאה:

Please enter a word or a sentence to check

לאחר מכן המחלקה תציג את הפלט הבא:

The word/sentence you entered: <ENTERED STRING>

The word/sentence written backwards: <ENTERED STRING BACKWARD>

אם הקלט הוא פלינדרום, יוצג הפלט הבא:

<ENTERED STRING> is a palindrome!

To quit click the EXIT button.

אם המחזורות אינה פלינדרום, יוצג הפלט הבא:

<ENTERED STRING > is NOT a palindrome!

To quit click the EXIT button.

בהצלחה!

פרק 5 דף עבודה מס' 5

ספר טלפונים אלקטרוני

מטרות

1. תרגול השימוש במערך חד-ממדי המכיל אובייקטים.
2. המשך העבודה עם ממשק המחלקה String.

מה עליכם לעשות?

עליכם לכתוב שלוש מחלקות:

- Entry – מגדירה אובייקט מסוג רשומה.
- PhoneBook – מגדירה אובייקט מסוג ספר טלפונים.
- Electronic Phone Book User Interface – EPhoneBookUI, מציגה למשתמש ממשק של ספר הטלפונים.

המחלקה Entry

מחלקה זו מגדירה אובייקט מסוג רשומה הכוללת פרטים על אודות אדם בספר טלפונים:
שם פרטי (firstName), שם משפחה (lastName) ומספר טלפון (phoneNumber).
ממשק המחלקה:

חתימת השיטה	תיאור השיטה
Entry (String firstName, String lastName, String phoneNumber)	שיטה-בונה היוצרת רשומה חדשה על פי הערכים שמועברים אליה
Entry (Entry otherEntry)	שיטה-בונה היוצרת רשומה חדשה הזזה (בכל שלוש התכונות) לרשומה המועברת אליה
void setFirstName (String firstName)	מעדכנת את השם הפרטי
void setLastName (String lastName)	מעדכנת את שם המשפחה
void setPhoneNumber (String phoneNumber)	מעדכנת את מספר הטלפון
String getFirstName()	מחזירה את השם הפרטי
String getLastName()	מחזירה את שם המשפחה
String getPhoneNumber()	מחזירה את מספר הטלפון
boolean equals (Entry otherEntry)	בודקת האם השם הפרטי ושם המשפחה של הרשומה שמפעילה את השיטה ושל הרשומה המועברת אליה זהים. אם השם הפרטי ושם המשפחה זהים מחזירה true , אחרת מחזירה false
String toString()	מחזירה מחרוזת המייצגת את הרשומה

שימו לב, השיטה equals(...) משווה בין שתי רשומות לפי התכונות: שם פרטי ושם משפחה בלבד. אם שתי התכונות הללו זהות, אז הרשומות זהות (גם אם מספר הטלפון ברשומות אינו זהה).

השיטה toString() מחזירה מחרוזת שנראית כך:

First name: <FIRST NAME>

Last name: <LAST NAME>

Telephone number: <TELEPHONE NUMBER>

המחלקה PhoneBook

מחלקה זו מגדירה ספר טלפונים הכולל מספר קבוע של רשומות. ממשק המחלקה הוא:

חתימת השיטה	תיאור השיטה
PhoneBook (<i>int</i> size)	שיטה-בונה היוצרת ספר טלפונים בגודל נתון
<i>boolean</i> addEntry (Entry entry)	מוסיפה רשומה לספר הטלפונים לפי המצב הנתון: אם הרשומה אינה קיימת – היא תתוסף למקום פנוי בספר. אם בספר הטלפונים כבר מופיעה רשומה של שם פרטי ושם משפחה הזהים לשמות המופיעים ברשומה שאותה רוצים להוסיף, אזי תוחלף הרשומה הקיימת ברשומה החדשה, באותו המקום בספר. אם הרשומה הוספה או עודכנה, השיטה מחזירה <i>true</i> , אחרת מחזירה <i>false</i>
<i>boolean</i> deleteEntry (Entry entry)	מוחקת רשומה מספר הטלפונים. אם הרשומה נמחקה מחזירה <i>true</i> , אחרת מחזירה <i>false</i>
String searchPhone (String firstName, String lastName)	מחזירה את מספר הטלפון המופיע ברשומה לפי שם פרטי ושם משפחה המועברים אליה. אם הרשומה קיימת, מחזירה את מספר הטלפון שלה, אחרת מחזירה <i>null</i>
String toString()	מחזירה מחרוזת המייצגת את ספר הטלפונים (כלומר את פרטי כל הרשומות שבספר הטלפונים)

השיטה toString() מחזירה מחרוזת שנראית כך:

First name: <FIRST NAME>

Last name: <LAST NAME>

Telephone number: <TELEPHONE NUMBER>

...

First name: <FIRST NAME>

Last name: <LAST NAME>

Telephone number: <TELEPHONE NUMBER>

...

First name: <FIRST NAME>

Last name: <LAST NAME>

Telephone number: <TELEPHONE NUMBER>

שימו לב :

1. מספר הרשומות שהספר יכול להכיל קבוע מראש (size), לכן ייתכן שנרצה להוסיף רשומה חדשה, אבל לא נוכל כי ספר הטלפונים כבר מלא. במקרה זה addEntry(...) תחזיר *false*.
2. בספר הטלפונים לא ייתכנו שתי רשומות של שם פרטי ושם משפחה זהים, לכן לא ניתן לשמור שני מספרי טלפון שונים בעבור אדם אחד.
3. השיטה deleteEntry (Entry entry) אחראית לכך שלא יישאר חור במערך. השיטה מוחקת חורים שנוצרו במערך בגלל שנמחקה רשומה, על ידי שינוי ההפניות מהמקום שנמחק ועד סוף המערך.
4. השיטה addEntry (Entry entry) מוסיפה רשומה חדשה לספר הטלפונים. אם רוצים להוסיף רשומה הזוהי לרשומה קיימת, כלומר שם המשפחה והשם הפרטי ברשומה זהים לאלו המופיעים באחת מהרשומות שכבר קיימות בספר, אזי השיטה תבטל את הרשומה הקיימת, ותוסיף את הרשומה החדשה במקום שבו הייתה הרשומה שזה עתה בוטלה. כך ניתן לשנות את מספר הטלפון של אדם שפרטיו כבר נמצאים בספר הטלפונים.

הערות :

1. במחלקה זו מומלץ לממש מספר שיטות עזר שאותן יש להגדיר כפרטיות *private*.
2. כאשר מממשים את שיטות המחלקה יש להשתמש בשיטות ההשוואה של המחלקה *.String*.

המחלקה EPhoneBookUI

המחלקה תציג למשתמש חלון פלט ובו התפריט הבא :

Your Electronic Phone Book

Please choose an operation code from menu:

1. Add an entry
2. Delete an entry
3. Find an entry
4. Print phonebook

To quit: click the Exit button

המשתמש יתבקש להקליד את בחירתו בחלון קלט. כותרת חלון הקלט יכולה להיראות כך :

Please select the operation code (see output window):

בהתאם לבחירת המשתמש יוצג פלט, על פי הפירוט דלהלן :

פעולה מספר 1: המשתמש יתבקש להכניס שלושה נתונים: שם פרטי, שם משפחה ומספר טלפון.
אם הרשומה החדשה הוספה לספר, תוצג ההודעה הבאה:

A new entry was added

אחרת (ספר הטלפונים מלא ולא ניתן להוסיף רשומה חדשה), תוצג ההודעה הבאה:

Sorry, your telephone book is full

פעולה מספר 2: המשתמש יתבקש להכניס שני נתונים: שם פרטי ושם משפחה.
אם הרשומה המבוקשת נמחקה מהספר, המשתמש יקבל את ההודעה:

Entry was deleted

אחרת, תוצג ההודעה הבאה:

Entry does not exist

פעולה מספר 3: המשתמש יתבקש להכניס שני נתונים: שם פרטי ושם משפחה, ולפיהם יתבצע החיפוש. אם הרשומה המבוקשת נמצאה, המשתמש יראה את פרטי הרשומה באופן הבא:

First name: <FIRST NAME>

Last name: <LAST NAME>

Telephone number: <TELEPHONE NUMBER>

אחרת, תוצג ההודעה הבאה:

Sorry, entry was not found

פעולה מספר 4: יוצגו הפרטים של כל הרשומות הכלולות בספר הטלפונים.

אם המשתמש בחר בפעולה שאינה מופיעה בתפריט, הוא יקבל את ההודעה הבאה:

Illegal operation code

הערות:

שימו לב, בכל פעם שמסתיים ביצוע של פעולה, עליכם להציג מחדש את תפריט הפעולות בחלון הפלט. לשם כך השתמשו בשיטה clear המופיעה במחלקה OutputWindow. התוכנית תיעצר כאשר המשתמש ילחץ על הכפתור Exit.
במחלקה זו מומלץ להגדיר מספר שיטות עזר, להן תוכלו לקרוא מתוך השיטה main(...).

בהצלחה!

פרק 5 דף עבודה מס' 6

ריבועי קסם*

מטרות

תרגול השימוש במערכים דו-ממדיים.

רקע

ריבוע קסם (Magic Square) בגודל n , הוא ריבוע שבו מסודרים המספרים מ-1 עד n^2 , כך שמתקיימים התנאים הבאים:

1. כל מספר מופיע פעם אחת בדיוק.
2. סכום המספרים בכל שורה, בכל עמודה ובכל אחד מהאלכסונים הראשיים זהה.

לדוגמה: ריבוע הקסם הפשוט ביותר הוא ריבוע בגודל 3×3 :

1

ריבוע קסם מתוחכם יותר הוא ריבוע קסם בגודל 3×3 :

6	1	8
7	5	3
2	9	4

מה עליכם לעשות?

בתרגיל זה עליכם לכתוב תוכנית המקבלת מהמשתמש מערך ריבועי (מטריצה) של מספרים שלמים, ובודקת האם המערך עונה על ההגדרה של ריבוע קסם.

קליטת הנתונים

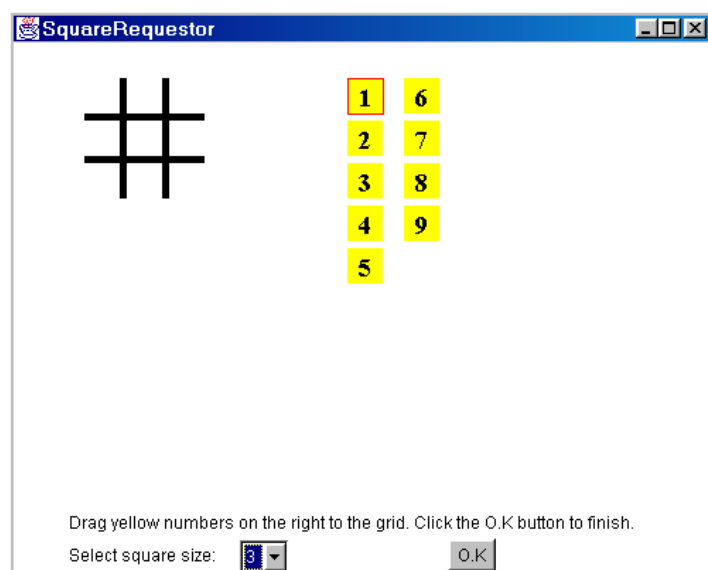
כדי לקבל מהמשתמש את הקלט, כלומר מערך ריבועי של מספרים שלמים, עליכם להשתמש במחלקה קיימת בשם `SquareRequestor`. אובייקט ממחלקה זו הוא חלון קלט, המאפשר קבלת מערך ריבועי מהמשתמש – המשתמש יכול לבחור את גודל המערך (ראו איור 1). על המשתמש לגרור בעזרת העכבר את המספרים המופיעים מימין, לתוך הריבוע המופיע משמאל. עם תום מילוי המערך, המשתמש ילחץ על הכפתור O.K כדי להמשיך בריצת התוכנית.

כדי להשתמש במחלקה זו עליכם להוסיף את הספרייה `MgSquare` אל הפרויקט שלכם. ספרייה זו מכילה את המחלקה `SquareRequestor` וכמה מחלקות נוספות, היוצרות ממשק גרפי (ראו איור 1). ממשק גרפי זה יאפשר לכם לקבל נתונים מהמשתמש בצורה נוחה וידידותית.

* תודה לפורום המתמטי של Drexel שהתירו לנו להשתמש בתוכנת ריבועי הקסם המלווה דף עבודה זה.
<http://mathforum.org>

את הספרייה MgSquare תוכלו למצוא בתיקייה :

C:\JCreator\MyProjects\Chap5\MagicSquare



איור 1 – אובייקט מטיפוס SquareRequestor

כדי ללמוד כיצד להשתמש במחלקה SquareRequestor, עיינו בממשק הבא :
ממשק המחלקה SquareRequestor :

חתימת השיטה	תיאור השיטה
SquareRequestor()	שיטה-בונה היוצרת עצם חדש מטיפוס SquareRequestor. העצם כולל מערך ריבועי של מספרים שלמים
<i>int</i> [][] getSquare()	מציגה בחלון גרפי ריבוע ללא ערכים (ראו איור 1), ממתינה שהמשתמש יכניס אליו נתונים ויאשר אותם ומחזירה את המערך הריבועי של האובייקט

שימו לב, העבודה עם אובייקט מסוג SquareRequestor דומה מאוד לעבודה עם מחלקת הקלט הגרפית InputRequestor שכבר הכרתם. לצורך הצגת פלט עליכם להשתמש בחלון הגרפי OutputWindow.

המחלקות שעליכם לכתוב

- MagicSquareChecker – מחלקת-שירות המאפשרת לבדוק אם מטריצה היא ריבוע קסם. כל השיטות במחלקה זו תהיינה שיטות סטטיות.
- MagicSquareGame – מחלקה הכוללת את השיטה main(...). במסגרת המחלקה, עליכם ליצור אובייקט מטיפוס SquareRequestor שבאמצעותו תקבלו מהמשתמש מערך ריבועי. לאחר מכן, על ידי שימוש בשיטות המחלקה MagicSquareChecker, עליכם לבדוק אם המערך שהתקבל הוא ריבוע הקסם.

המחלקה MagicSquareChecker

מחלקה זו מאפשרת לבדוק אם מערך ריבועי הוא ריבוע קסם. כדי לערוך בדיקה מעין זו השתמשו במחלקה ArrayUtil שכבר כתבתם.

עליכם לממש את הממשק הבא :

תיאור השיטה	חתימת השיטה
מחזירה <i>true</i> אם המערך שהועבר הוא ריבוע קסם. אינה בודקת אם המערך שהועבר הוא מערך ריבועי	<i>static boolean</i> isMgSquare (<i>int</i> [][] mat)

הערה : בהגדרת המחלקה מומלץ להשתמש במספר שיטות עזר (*private*).

המחלקה MagicSquareGame

מחלקה זו תכלול את השיטה *main(...)*, ועליה לבצע את הפעולות הבאות :

1. המחלקה תציג חלון פלט (השתמשו במחלקה *OutputWindow*), ובו ההודעה הבאה :
Try to create a magic square
2. יוצג חלון קלט מטיפוס *SquareRequestor* (ראו איור 1), שיאפשר למשתמש להרכיב מערך ריבועי.
3. המחלקה תקבל את ריבוע המספרים שהמשתמש הרכיב (ראו את הממשק במחלקה *SquareRequestor*) ותבדוק האם המספרים יוצרים ריבוע קסם. אם כן, תוצג בחלון הפלט ההודעה :

YES! You got a Magic Square!

To end game click Exit button

אם המספרים אינם יוצרים ריבוע קסם, תוצג ההודעה הבאה :

NO! This is not a Magic Square.

To end game click Exit button.

שימו לב, שכל עוד המשתמש לא בחר בכפתור *Exit* של חלון הפלט, הוא יוכל להמשיך ולשחק במשחק. לשם כך עליכם להשתמש בלולאה אינסופית לבקשת המערך הריבועי של האובייקט. לולאה זו תופסק אוטומטית כאשר המשתמש ילחץ על הכפתור *Exit* של חלון הפלט.

בהצלחה!

פרק 5 דף עבודה מס' 7

תוכנת התיעוד javadoc

מטרות

1. הגדרת תוכנת התיעוד בתוך סביבת העבודה (חד פעמי!).
2. הכרת תוכנת התיעוד javadoc.
3. לימוד כללי התיעוד המאפשרים להפעיל את מנגנון התיעוד.

רקע

בדף עבודה זה נלמד להשתמש במנגנון התיעוד javadoc. מנגנון זה מאפשר לייצר דפי HTML סטנדרטיים, בעלי אופי אחיד, המכילים את הממשק למשתמש בפורמט המוכר לכם מדפי ה-API שאתם עבדתם. דפי הממשק מיוצרים על פי ההערות שהוספתם לקוד, ובתנאי שההערות נכתבו בפורמט הנכון. התיעוד האחיד מאפשר שימוש נוח במחלקות שונות על ידי משתמשים רבים.

מה עליכם לעשות?

1. בחרו קוד מחלקה שכתבתם (אפשר לבחור את קוד תוכנית המלבן – Rectangle). מומלץ לא לבחור במחלקה הכוללת שיטת main(...) (בלבד).
2. תעדו את הקוד באופן מלא לפי ההנחיות הבאות:
 - כל הערה המתעדת מידע, הנחוץ למי שישתמש במחלקה (משתמש חיצוני), תתחיל ב-/**/ ותסתיים ב-*/.
 - הערות פנימיות הנכתבות לצורך המתכנת יתחילו ב-// או ב-/*.

תחימת ההערה בין הסימנים /**/ ו-/*/ מסמנת למנגנון התיעוד להעתיק את ההערה הזו מגוף הקוד אל דף הממשק. לכן, חשוב שכל הערה הנוגעת לממשק המחלקה (ורק הערה כזו) תיכתב בין הסימנים /**/ ו-/*/.

מקובל גם לפתוח כל שורה חדשה בהערה בסימן *. סימן זה הוא תוספת סגנונית בלבד, ואינו הכרחי לפעולתו של מנגנון התיעוד.

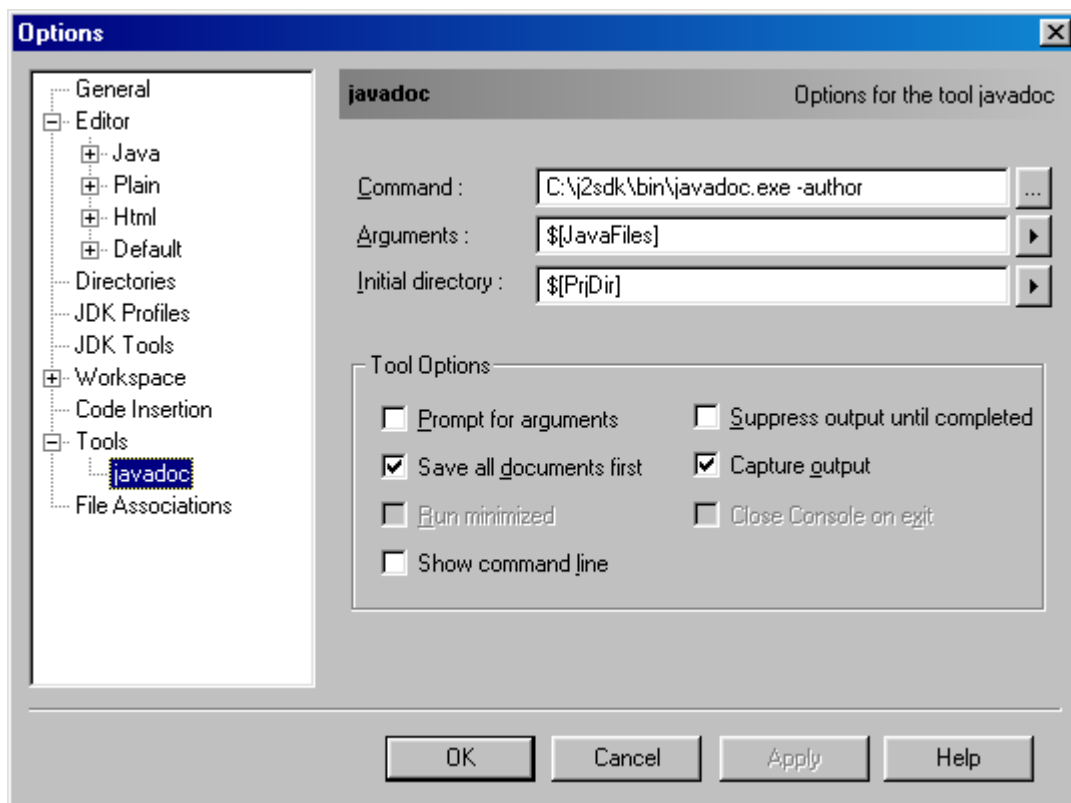
הגדרת תוכנת התיעוד בתוך סביבת העבודה

לפני שנוכל לייצר את התיעוד המתאים, עלינו להגדיר כראוי באופן חד-פעמי את תוכנת התיעוד בתוך סביבת העבודה. עקבו במדויק אחר ההנחיות ובצעו את כל השלבים הבאים כסדרם, גם אם אינכם מבינים מה בדיוק אתם מבצעים. הקישור הוא טכני בעיקרו, אך חובה לבצע אותו בפעם הראשונה במלואו ובמדויק:

1. העלו את סביבת הפיתוח.
2. מהתפריט Configure בחרו באפשרות Options.
3. בתיבה השמאלית המופיעה בחלון Options, סמנו את האפשרות Tools.
4. לחצו על הכפתור New המופיע מימין, ובחרו באפשרות Program.
5. בחלון Open אתרו את התיקייה j2sdk.

6. בתיקייה j2sdk, בחרו בתיקייה bin, ולאחר מכן בחרו את הקובץ של תוכנת התיעוד javadoc. הקליקו עליו פעמיים.
7. בעץ המופיע בתיבה משמאל סמנו את האפשרות (Tools\Javadoc) javadoc.
8. בשדה Commands המופיע בתיבה הימנית, הוסיפו את הפקודה -author. שימו לב: יש להשאיר רווח בין הכתוב הקיים לתוספת.
9. לחצו על החץ המופיע מימין לשדה Arguments ובחרו באפשרות Project Java Files.
10. לחצו על החץ המופיע מימין לשדה Initial directory ובחרו באפשרות Project directory.
11. ודאו כי האפשרויות Save all documents first ו-Capture output מסומנות. אם אפשרויות אחרות מסומנות, בטלו אותן (ראו איור להלן).
12. לחצו על Apply כדי להחיל את התצורה החדשה על התוכנה.
13. לחצו על OK לסיום.

כעת תוכלו להפעיל את תוכנת התיעוד מתוך סביבת הפיתוח JCreator.



לעתים הגדרת javadoc משנה את הפונט של קובצי ג'אווה. דרך הטיפול בבעיה זו מתוארת במדריך הטכני סעיף 5.3.

הפעלת תוכנת התיעוד

1. כאשר הפרויקט פתוח, בחרו מתפריט התוכנה JCreator את הפקודה javadoc הנמצאת בתפריט Tools. לחיצה אחת על פקודה זו תפעיל את מנגנון התיעוד javadoc עבור כל קובצי הקוד המופיעים בפרויקט.

2. לא להבהל! הפעלת ה-javadoc גורמת למהדר לכתוב שורות רבות בחלון הקומפילציה. שורות אלו מעידות על התהליך המתרחש. אם הן מסתיימות בהודעה התקינה שהתהליך הושלם (Process completed) אזי ניתן להתעלם מהן.
3. כדי לראות את תוצאות עבודתכם, אתרו את תיקיית הפרויקט שלכם (בעזרת תוכנת WindowsExplorer או MyComputer). בתוך תיקיית הפרויקט תמצאו כמה קובצי HTML שיצר מנגנון ה-javadoc. הקובץ העיקרי שבו תשתמשו הוא הקובץ הנושא את שם קובץ הג'אווה שאותו תיעדתם; הסיומת שלו היא 'html.' (התעלמו משאר הקבצים הנושאים את שם הקובץ המקורי, אך להם סיומות אחרות. ניתן למחוק אותם).
4. פתחו את קובץ ה-'html'. עברו על הקובץ והשוו בין התיעוד שנוצר לתיעוד שכתבתם אתם במחלקה.
5. עד כה, ראינו את פעולתו הבסיסית של מנגנון javadoc. למנגנון זה אפשרויות רבות, ואנו נציג כאן שלוש מתוכן:

א. **שם כותב המחלקה** – בתוך גוף ההערה המתעדת את המחלקה הוסיפו את הפקודה `@author <yourName>`, (לאחר ההערה שכבר כתבתם).
לדוגמה:

```
/**
 * This class represents a...
 * @author Zrubavela Yakinton.
 */
שימו לב לסימן @, המופיע לפני הפקודה author. סימן זה מופנה אל מנגנון
ה-javadoc, ומורה לו להתייחס לאינפורמציה המופיעה לאחר ה-@ ולהוציאה
אל דף התיעוד.
```

ב. **פרמטרים של שיטה** – כדי להוסיף בתיעוד פירוט של הפרמטרים שמקבלת שיטה, יש להשתמש בפקודה `<paramName> <explanation>`.
לדוגמה, את השיטה:

```
void setX(int x)
```

נוכל לתעד על ידי:

```
/**
 * changes the x coordinate to the given value.
 * @param x – the new value for the x coordinate.
 */
ג. ערך החזרה – כדי להוסיף את פירוט ערך החזרה של שיטה, נשתמש בפקודה
@return.
לדוגמה, את השיטה:
```

```
int getY()
```

נוכל לתעד על ידי:

```
/**
 * returns the y coordinate.
 * @return the y coordinate.
 */
```


6. הוסיפו לתיעוד המחלקה את שמכם בעזרת שימוש בפקודה @author. הדרו שנית את קוד המחלקה, ושימרו את קובץ הקוד.

7. הפעילו מחדש את תוכנת התיעוד, ופתחו מחדש את קובץ ה-html' (עתה יהיה זה הקובץ החדש שנוצר). מהם ההבדלים בין קובץ זה לקובץ הקודם שיצרתם?

8. נסו להפעיל את הפקודות השונות של javadoc ולראות כיצד הן משפיעות על קובץ הממשק שיווצר. מידע נוסף על תוכנת התיעוד javadoc תוכלו למצוא באתר חברת סאן:

<http://java.sun.com/j2se/javadoc/index.html>

כעת, כשאתם יודעים לעבוד עם ממשקים למשתמש וגם לייצר אותם, הקפידו לתעד כל מחלקה שתכתבו על פי הכללים החדשים. כך תוכלו לאפשר למשתמשים אחרים לעבוד עם המחלקות שכתבתם בעזרת דפי ה-HTML, מבלי להזדקק לקוד המקור.

סיכום

בדף זה הכרנו את מנגנון התיעוד javadoc. על ידי מנגנון זה, אפשר באופן פשוט ויעיל לייצר ממשקים בעבור מתכנתים נוספים המשתמשים במחלקות שכתבתם. העבודה עם javadoc תאפשר לכם ליישם את עקרונות האנקפסולציה ועקרונות העבודה עם ממשקים, שהם מעמודי התווך של תכנות מונחה עצמים.

תקציר התגיות של javadoc

בטבלה שלהלן מצורפות כמה תגיות שמורות של מנגנון התיעוד javadoc. את הפירוט המלא של התגיות תוכלו למצוא באתר של חברת סאן: www.java.sun.com.

התגית	משמעות התגית
@author	שם מחבר המחלקה
@param	פרמטר המועבר לשיטה
@return	הערך שמחזירה השיטה
@see	הפנייה למחלקה אחרת הקשורה למחלקה זו

מהf3חה!

