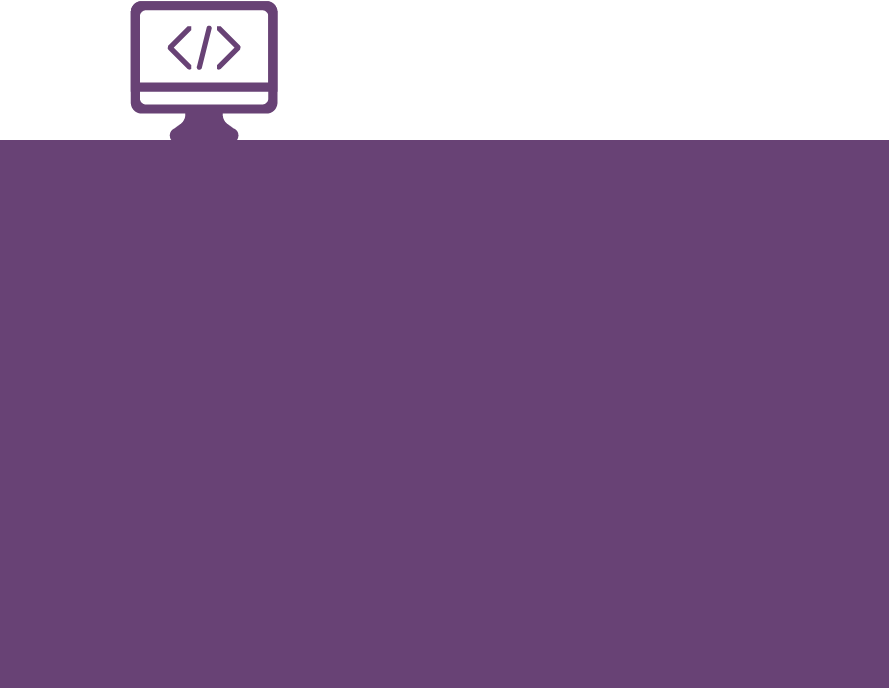
## שער 1: יסודות מדעי המחשב

מדעי המחשב



|  |  |
| --- | --- |
| הגוף המבצע: | אורט ישראל |
| ראש המו"פ: | מיכל בירן |
| ראש צוות הפיתוח: | ד"ר משה טלסניק |
| ריכוז הפרויקט: | ירדן מוסינזון-פינטו ודניאל סמוקובליה |
| כתיבה: | שרה פונק ושמוליק שוורץ (שער I), דפנה מינסטר ובנימין ברנד (שער II) |
| ייעוץ מדעי: | ד"ר גרא וייס, מיכל שמש |
| ייעוץ דידקטי: | ירדן מוסינזון-פינטו |
| עריכה לשונית ומגדרית: | אורלי ניטיס יעקובי |
| עיצוב גרפי ואינטראקטיבי: | עמית גל, לאה כהן, חגית מתוק, ולרי פקר |
| הפקה ועימוד: | אוליבר ריקלין, מיכל שי |
|  |  |



המינהל לתקשוב, טכנולוגיה ומערכות מידע

משרד החינוך

© כל הזכויות שמורות למשרד החינוך

###### דבר המפמ"ר

שפת מדעי המחשב היא שפה במרחב השפות שכולל את השפה בה הוא קורא וכותב (השפה שלנו); את השפה הזרה (השפה שלהם); את שפת המדע (המתמטיקה); את שפת האמנות והגוף ואת שפת הטכנולוגיה – מדעי המחשב. שפה המתארת תהליכים, מבנים וקשרים בין טכנולוגיות.

החשיבות של מדעי המחשב כענף מרכזי גדלה והיא משפיעה על כל היבט של חיינו. תלמיד שפותר בעיה באמצעות תכנית מחשב משתמש במיומנויות הפשטה, ביכולת לנתח ולממש פתרון בעיות/ משימות בעזרת כתיבה והפעלה של תכניות מחשב ובהצגת תכניות למימושן. הוא צריך להיות מסוגל לייצג בעיה/משימה מתחומים שונים (פיסיקה, מתמטיקה, כימיה, ניהול, בעיות/משימות ממדעי החברה והרוח) באמצעות ייצוג/מודל או אלגוריתם שניתן לתרגמו לתכנית מחשב.

השגת מכלול היעדים דברי אינה משימה קלה כלל ועיקר אבל תחום מדעי המחשב מהווה פלטפורמה להשגתם ומכלול היעדים שמניתי הם תוצר טבעי של תהליכי הלמידה בהוראת מדעי המחשב.

ד"ר אבי כהן

מפמ"ר מדעי המחשב וממונה מגמות (טכנולוגיות עתירות ידע)

משרד החינוך

###### פרק 1: מבוא

#### ממשחק לאלגוריתם

**רגע חושבים!**

מדוע לדעתכם כולנו נהנים ממשחק? מה הקשר בין משחקים למדעי המחשב?

כאשר אנו משחקים עם חברים או לבדנו, אנחנו מתכננים אסטרטגיה (תוכנית) שתוביל אותנו לנצח במשחק. בתחום מדעי המחשב נמצאת תיבת כלים גדולה לתכנון ולבניית אסטרטגיות כאלה. אפשר לראות במדעי המחשב את שפת הטכנולוגיה שבה מיושמים הידע, הכלים והמיומנויות הללו. השפה מתארת תהליכים, מבנים וקשרים בין טכנולוגיות כדי להרחיב את יכולת האדם הנבון ולפתור בעיות מעשיות.

את תהליך הלמידה במדעי המחשב נתחיל במשחקים:

1. **היכנסו** לקישור הבא, והורידו את המשחקים על פי דף ההוראות.
2. באתר זה תוכלו לשחק בארבעת המשחקים הבאים:
   * **מגדלי האנוי**
   * **משבר קבוצתי**
   * **היגיון של צפרדעים**
   * **מטבעות זהב**
3. עבור כל משחק **פתרו** את המשימות המופיעות בעמודים הבאים, המשימות יעזרו לכם בהבנת המשחקים.



##### משימה 1: משחק "מגדלי האנוי"

במשחק זה יהיה עליכם להעביר את כל הטבעות מעמוד המקור אל עמוד היעד תוך שמירה על הכלל הבא: אסור שטבעת גדולה תוצב מעל טבעת קטנה ממנה. הכלל נוגע לכל שלושת עמודי המשחק.

### חלק א'

1. **שחקו** במשחק. בחרו באפשרות של 3 טבעות במוט המקור.
2. האם הצלחתם במשחק? כמה העברות של טבעות ביצעתם בסך הכול?
3. **שחקו** במשחק. בחרו באפשרות של 5 טבעות במוט המקור.
4. האם הצלחתם במשחק? כמה העברות של טבעות ביצעתם בסך הכול?
5. **השיבו** על השאלות הבאות:
   * מהי **נקודת המוצא** במשחק?
   * מהו **היעד**?
   * מהו **רצף הפעולות** להשגת היעד במשחק של 3 טבעות?
   * האם ייתכן מצב של **פעולות** **מיותרות**?

לפני שמתחילים לשחק, אנו מתכננים לעצמנו כיצד נצליח לעשות זאת. במילים אחרות, אנו רושמים לעצמנו, בכתב או בראש, רצף של פעולות להשגת המטרה ולפתרון הבעיה.

**פתרון**

נתאר את המשחק:

* **נקודת המוצא:** 5 טבעות מוצבות על מוט המקור.
* **היעד:** להעביר את כל הטבעות אל מוט המטרה, תוך שמירה על הכלל.
* כדי לפתור את הבעיה יש לתאר רשימה של הוראות בצורה הבאה: "העבר את הטבעת העליונה ממוט מסוים אל מוט אחר".

למשל, הפתרון עבור 3 טבעות הוא:

העבירו טבעת 1 מעמוד 0 אל עמוד 2

העבירו טבעת 2 מעמוד 0 אל עמוד 1

העבירו טבעת 1 מעמוד 2 אל עמוד 1

העבירו טבעת 3 מעמוד 0 אל עמוד 2

העבירו טבעת 1 מעמוד 1 אל עמוד 0

העבירו טבעת 2 מעמוד 1 אל עמוד 2

העבירו טבעת 1 מעמוד 0 אל עמוד 2

### חלק ב'

1. **כתבו** פתרון דומה עבור 5 טבעות
2. **השיבו** על השאלות הבאות:
3. כיצד הייתם מנחים את חבריכם לכיתה לפתור את הבעיה המוצגת במשחק?
4. האם לדעתכם, כל תלמידי הכיתה הנחו את החברים בדיוק כמו שאתם הנחיתם כדי שיצליחו בפתרון החידה? הסבירו את תשובתכם.

##### משימה 2: משבר קבוצתי

1. **תכננו** את המהלכים לפתרון הבעיה לפני שהשעון מתחיל לרוץ על מנת להצליח במשימה בניסיון יחיד. האם הצלחתם לפתור את המשבר הקבוצתי בפעם הראשונה ששיחקתם?
2. **השיבו** על השאלות הבאות:
   * מהי **נקודת המוצא** במשחק?
   * מהו **היעד**?
3. **כתבו** מהם הצעדים בדרך להצלחה. הסבירו במילים שלכם מה היה השיקול המרכזי שלכם בדרך לפתרון.

**פתרון**

* **נקודת המוצא**: חמישה בעלי חיים ניצבים בלילה בצד ימין של תהום. לרשותם מנורת לילה.
* **היעד**: יש להעביר את כל הקבוצה על קורת העץ לצד השמאלי של התהום. הפעם מציבים לנו אתגר נוסף: לא רק להעביר את חברי הקבוצה בעזרת המנורה, אלא גם לעשות זאת בפרק זמן מוגבל.



##### משימה 3: היגיון של צפרדעים

1. **השיבו** על השאלות הבאות:
   * מהי **נקודת המוצא** במשחק?
   * מהו **היעד**?
2. **כתבו** כיצד ניתן היה להצליח במשימה בניסיון הראשון? תארו במילים שלכם מה היה השיקול המרכזי שלכם בפתרון הבעיה.

**פתרון**

במשחק זה, נקודות המוצא והיעד הן:

* **נקודת המוצא**: בצד ימין שורה של חמש צפרדעים ירוקות ובצד שמאל שורה של חמש צפרדעים כחולות.
* **היעד**: צריך להעביר את כל הצפרדעים הירוקות לצד ימין של המסך ואת כל הצפרדעים הכחולות לצד שמאל בהתאם לכללים.

##### משימה 4: מטבעות זהב

נמצאת בידיכם תיבה המכילה תשעה מטבעות זהב, וידוע שאחד מהם מזויף. משקלו של המטבע המזויף קטן ממשקל המטבע שעשוי זהב.

לרשותכם עומדים מאזניים.



**כתבו** רצף פעולות למציאת המטבע המזויף שבו יבוצעו לכל היותר 3 פעולות שקילה.

הערה: ניתן להשתמש במשפטי תנאי המוכרים לכם מחיי היום-יום.

לדוגמה: נבחר שני מטבעות ונניח כל מטבע על כף המאזניים.

אם המאזניים אינם מאוזנים, המטבע המזויף יימצא בכף שעלתה למעלה.

**כתבו** פעולות למציאת המטבע המזויף באמצעות 2 פעולות שקילה בלבד.

**מה הקשר בין חידות ההיגיון לבין מדעי המחשב?**

מדעי המחשב הם מדע העוסק בפתרון בעיות רבות ומגוונות בתחום המשחקים, החישובים, האפליקציות ועוד. בספר זה תכירו חלק מהבעיות בתחום, חלקן פשוטות ואחרות מורכבות מאוד. את הפתרונות אפשר בהמשך ליישם ולהפעיל בעזרת מכונה שנקראת "מחשב".

בפרק זה נכיר את הנושא שבו נעסוק במהלך כל לימוד היחידה: פתרון בעיות אלגוריתמיות. במרכזו יעמוד המושג **אלגוריתם**, מושג מרכזי וחשוב במדעי המחשב, שמשמעו **תכנון פתרון לבעיה** **שאפשר לתרגמו לתוכנית מחשב**.

**רגע חושבים!**

מהו אלגוריתם? ננסה להסביר.

**אלגוריתם** הוא רישום ראשוני (סקיצה) לפתרון בעיה במספר צעדים סופי. אלגוריתם מורכב תמיד מרצף פעולות חד-משמעיות ואפשריות לביצוע, אשר סדר ביצוען מוגדר היטב. לאורך הלימוד, נקפיד לבדוק את האלגוריתם שלנו כך שהוא עומד בשלושה תנאים:

פעולות האלגוריתם מבוצעות לפי סדר הופעתן כל עוד לא נאמר אחרת.

כאשר ביצוע פעולה אחת מסתיים, ממשיכים לפעולה הבאה עד שנגמרות הפעולות.

לרוב נעדיף שביצוע האלגוריתם יסתיים אחרי מספר סופי של פעולות.

 מה מקור השם אלגוריתם?

המונח "אלגוריתם" נגזר ככל הנראה משמו של המתמטיקאי מוחמד אל-חואריזמי, שהשתבש לאל-גואריזמי. אל-חואריזמי חי במאה התשיעית לספירה באזור חואריזם (היום במדינת אוזבקיסטן). הוא היה הראשון שניסח את הכללים המשמשים אותנו עד היום לביצוע ארבע פעולות החשבון הבסיסיות. במאה ה-14 החלו מתמטיקאים להשתמש במונח "אלגוריתם" ככינוי לרצף פעולות מתמטיות.

כאלו הם למשל אלגוריתם להכפלת שני מספרים (כפל ארוך), אלגוריתם להעלאה בחזקה, אלגוריתם למציאת המחלק המשותף הגדול ביותר של שני מספרים שלמים חיוביים (האלגוריתם של אוקלידס), ועוד.

במדעי המחשב אלגוריתמים מתוכננים לפתור **בעיה אלגוריתמית** – בעיה שמקיימת את התכונות הבאות:

* מוגדרת נקודת מוצא ותנאי התחלה.
* מוגדר תנאי הסיום – המטרה שאותה רוצים להשיג.

**שימו לב!** לבעיה נתונה אפשר להציג מספר פתרונות, כלומר אלגוריתמים שונים.

##### בהמשך הלימודים נחזור לנושא האלגוריתם ונרחיב אותו.



##### משימה 5: יעילות אלגוריתם

**השיבו** על שתי השאלות הבאות:

1. האם במשחק "לא לערבב את הצבעים" אפשר להעביר את הצבעים הלוך וחזור ללא התקדמות? האם העברה זו משפיעה על פתרון הבעיה?
2. האם במשחק "משבר קבוצתי" כדאי להעביר את חברי הקבוצה הלוך וחזור ללא התקדמות?

**פתרון**

1. אפשר לפתור את המשחק גם עם צעדים מיותרים, ועדיין נגיע לפתרון הבעיה.
2. במקרה זה, לא כדאי להעביר את חברי הקבוצה הלוך וחזור ללא צורך, כיוון שזהו בזבוז של זמן במשחק ועלול למנוע פתרון לבעיה.

### יעילות אלגוריתם

ראינו כי ניתן להשוות בין אלגוריתמים שונים לאותה בעיה. בהמשך הלימודים נשווה בין אלגוריתמים גם על פי מידת היעילות שלהם.

##### העולם כולו עצמים

**רגע חושבים!**

כיצד ילדים לומדים על העולם שסביבם?

כאשר ילד לומד ומגלה את העולם, הוא מבחין בעצמים שונים הקיימים בו. באופן טבעי, אנחנו תופסים את העולם כמורכב מעצמים. בחיי היום-יום אנו נתקלים בעצמים רבים שמאופיינים בתכונות ופעולות שונות. עם הזמן אנו מאפיינים עצמים בלי לשים לב לכך, ומבלי לציין זאת במפורש. אפיונו של העצם על פי התכונות והפעולות שלו מובן מאליו עבורנו, כיוון שזהו תהליך טבעי ואינטואיטיבי.

נסתכל על שתי דוגמאות לעצמים, ועל כמה תכונות ופעולות לדוגמה:

* **טלפון סלולרי**

תכונות: כמות המקום בזיכרון, מצב הסוללה, מראה המסך, תצוגה גרפית.

פעולות: קבלת שיחה נכנסת, חיוג, גלישה באינטרנט, הורדת אפליקציות.

* **צפרדע**

תכונות: שם מדעי, מין, אזור מחיה.

פעולות: קפיצה, אכילה, תקשורת באמצעות קולות.

ניתן להתייחס לכל פריט בעולם – חי, צומח, דומם, מופשט – כאל עצם שניתן לאפיין באמצעות תכונותיו, ובדרך כלל גם באמצעות פעולותיו.

##### 

##### משימה 6: דוגמאות לעצמים

1. **כתבו** דוגמאות לשני עצמים, ופרטו עבור כל אחד מהם שתי תכונות ושתי פעולות.
2. האם המחשב הוא עצם? אם כן, מהן התכונות והפעולות שלו?



##### משימה 7: משחק זיכרון

1. **היכנסו** לכתובת הבאה: <http://games.yo-yoo.co.il/games_play.php?game=4521>
2. **שחקו** באחד ממשחקי הזיכרון. ניצחתם? מצוין.
3. **בחרו** משחק אחד מבין משחקי הזיכרון שראיתם באתר.
4. **השיבו** על השאלה: מהם העצמים במשחק? ממה מורכב המשחק?
5. **כתבו** את המאפיינים של הקלפים כעצמים:
   * מהן התכונות של כל קלף? מה משותף לכל הקלפים ומה שונה?
   * אילו פעולות ניתן לבצע עליהם?

משחק הזיכרון מורכב מקלפים, שהם העצמים במשחק.

תכונות: כל קלף הוא מרובע בעל שני צדדים. בצד אחד שלו יש תמונה, למשל בעל חיים, ובצד השני רקע קבוע ללא תמונה.

פעולות: פתיחת קלף, סגירת קלף, השוואה בין שני קלפים.

נחזור שוב אל האתר. בחרו משחק אך אל תתחילו לשחק עדיין. מה קורה בתחילת התהליך, כאשר אנחנו בוחרים לשחק במשחק מסוים?

זהו השלב שבו מתבצע אתחול (Initialization) של המשחק – הופעת הקלפים וסידורם על מנת שנוכל להתחיל לשחק.

##### משימה 8: כתיבת אלגוריתם

נתון משחק זיכרון ובו 16 קלפים.

1. **כתבו** אלגוריתם המתאר **תור של שחקן** במשחק.
2. האם האלגוריתם לתור של שחקן מתאים למשחק זיכרון ובו 40 קלפים?
3. האם האלגוריתם מתאים למספר חיובי לא ידוע של קלפים?
4. נתון משחק זיכרון עם מספר חיובי לא ידוע של קלפים. **השלימו** את האלגוריתם למהלך המשחק:

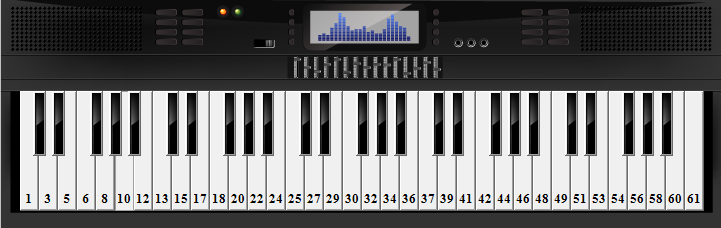
כל עוד \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

מתבצע **תור של שחקן**

בפרק זה הכרנו בעיות אלגוריתמיות והתמודדנו עם ניסיונות לכתוב אלגוריתמים כדי לפתור בעיות. עשינו זאת בעיקר דרך משחקים. למדנו שאלגוריתם הוא רצף פעולות לפתרון בעיה. במדעי המחשב אנו לומדים לפתור בעיות. אם ברצוננו לתת למחשב לבצע את האלגוריתם עבורנו, עלינו לתרגם אותו לתוכנית מחשב.

##### משימה 9: האורגן שלנו

**שימו לב!** יש להפעיל את הרמקולים במחשב לצורך משימה זו.



לרשותכם אורגן (ראו בציור מעלה). ניתן להנחות את האורגן לבצע את הפעולות הבאות:

1. לנגן כל אחד מהתווים הבאים: דו, רה, מי, פה, סול, לה, סי (התווים ממוספרים מ-1 ועד 61 כולל). דו הוא תו מספר 25, רה מספר 27, מי מספר 29, פה מספר 30, סול מספר 32, לה מספר 34 וסי מספר 36.
2. להקליט – לשמור נגינה של תו אחד או יותר. כאשר מקליטים לא שומעים את הצלילים.
3. להשמיע את ההקלטה.
4. למחוק את סדרת הצלילים שהוקלטה .

אפשר להציג את תוצאת הפעולות, כלומר "להריץ את התכנית", על גבי מסך המחשב. התוצאה הנראית לעין (או נשמעת לאוזן במקרה שלנו) של הרצת התוכנית נקראת **פלט.**

לדוגמה, נתון האלגוריתם הבא:

**התוכנית**

אורגן נגן את התו מי ואחר כך נגן את התו סול

אורגן הקלט את התו פה ואחר כך הקלט את התו סול ואת התו דו

אורגן השמע הקלטה

אורגן מחק הקלטה

אורגן הקלט את התו מי פעמיים

אורגן השמע הקלטה



משימה 10: כתיבת התכנית

כדי להשתמש במחלקהOrgan עליכם לייבא את החבילה unit4.organLib לכל תוכנית שבה תשתמשו במחלקה זו באמצעות הפעולה import.

כדי שנוכל לטפל באורגן בתוכנית עלינו ליצור אותו ולתת לו שם.

Organ myOrgan = new Organ() ;

בכל פעם שנרצה שהאורגן יבצע פעולה נפנה אליו באמצעות שמו ואחר כך נרשום נקודה (.) ושם הפעולה לביצוע. בסוף כל פעולה נוסיף את הסימן ;

myOrgan.play(29);

myOrgan.play(32);

myOrgan.add(30;(

myOrgan.add(32);

myOrgan.add(25);

myOrgan.play();

myOrgan.Clear();

myOrgan.add(3;(

myOrgan.add(3);

**הפעילו** את התכנית במחשב.

הצגת הקלף נעשית באמצעות הפעולה showCard

פתיחת הקלף (כמו במשחק הזיכרון ששיחקנו) נעשית באמצעות הפעולה openCard.

מה לדעתכם יהיה הפלט של התוכנית?

### משימה 11: שינויים באלגוריתם

ערכנו שינוי באלגוריתם הנתון במשימה 10:

אורגן נגן את התו מי ואחר כך נגן את התו סול

אורגן הקלט את התו פה ואחר כך הקלט את התו סול ואת התן דו

אורגן השמע הקלטה

אורגן הקלט את התו מי פעמיים

אורגן השמע הקלטה

1. **כתבו** בקצרה מה ההבדל בין האלגוריתמים.
2. האם המנגינה שנשמעה בעת ביצוע האלגוריתם בגרסה הנוכחית תהיה זהה למנגינה שנשמעה בעת ביצוע הגרסה הראשונה של האלגוריתם? **נמקו** את תשובתכם.

חלק מהפעולות שיכול עצם מטיפוס אורגן לבצע, המוגדרות על ידי המחלקה Organ, מתוארות בטבלה הבאה שנקראת ממשק למשתמש:

|  |  |
| --- | --- |
| Organ() | הפעולה יוצרת עצם מסוג אורגן. |
| play( מספר התוnum) | הפעולה משמיעה את התו שמספרו num.  num הוא מספר בין 1 ל-61. |
| add(מספר התו num) | הפעולה מקליטה את התו שמספרוnum , כלומר מוסיפה אותו לאוסף התווים המוקלטים.  התו המוקלט אינו מושמע.  num הוא מספר בין 1 ל-61. |
| play() | הפעולה משמיעה את כל התווים המוקלטים |
| clear() | הפעולה מוחקת את כל התווים המוקלטים, מסיום המחיקה האחרונה ועד כה. |

משימה 12: פיתוח אלגוריתם וכתיבת תכנית

1. **כתבו** אלגוריתם המקליט באורגן מנגינה.
2. **השמיעו** את המנגינה שהקלטתם.
3. **כתבו** תוכנית מחשב המממשת את האלגוריתם.
4. **הריצו** את התוכנית, אם יש צורך בתיקונים עשו זאת.

##### משימה לסיכום הפרק – כותבים ביחד

לרשותכם צב פלאים שאותו ניתן להנחות:

1. לנוע על המסך בקווים ישרים ולהגדיר בכמה צעדים הוא מתקדם.
2. לפנות בזווית ימינה או שמאלה ולהגדיר את גודלה.
3. לצייר בעזרת הזנב.

על מנת לצייר על המסך, זנב הצב צריך להיות למטה.

על מנת להזיז את הצב ללא ציור, הזנב צריך להיות למעלה.

:

משימה 13: הצב המצייר מלבן



**כתבו** אלגוריתם המצייר בעזרת הצב מלבן שאורכו 100 צעדים ורוחבו 40 צעדים.

מצבו ההתחלתי של הצב: עומד במרכז הדף, פניו צפונה ▲ וזנבו מורם.

* הצב צריך להוריד את הזנב.
* הצב צריך להתקדם קדימה 100 צעדים, ואחר כך לפנות עם כיוון השעון בזווית של 90 מעלות.
* הצב צריך להתקדם קדימה 40 צעדים, ואחר כך לפנות עם כיוון השעון בזווית של 90 מעלות.
* הצב צריך להתקדם קדימה 100 צעדים, ואחר כך לפנות עם כיוון השעון בזווית של 90 מעלות.
* הצב צריך להתקדם קדימה 40 צעדים, ואחר כך לפנות עם כיוון השעון בזווית של 90 מעלות.

האם לדעתכם זהו הפתרון היחיד למשימה שקיבלתם?

כתבו לפחות שני אלגוריתמים נוספים לפתרון הבעיה.

כתבו תוכנית מחשב אשר מממשת את האלגוריתם– תצייר מלבן באמצעות צב הפלאים.

לרשותכם מחלקה קיימת בשם Turtle שממנה ניתן ליצור עצמים מסוג צב. באמצעות הצב ניתן לצייר על משטח הגרפי. כך נראה עצם מסוג צב לאחר שיצרנו אותו:



כדי להשתמש במחלקה עליכם לייבא את החבילה unit4.turtleLib לכל תוכנית שבה תשתמשו בצבים באמצעות הפעולה import.

חלק מהפעולות שיכול עצם מטיפוס צב לבצע, המוגדרות על ידי המחלקה Turtle, מתוארות בטבלה הבאה שנקראת ממשק למשתמש:

|  |  |
| --- | --- |
| Turtle() | הפעולה הבונה עצם מסוג צב הממוקם במרכז, פניו פונים צפונה וזנבו מופנה כלפי מעלה. |
| moveForward(מספר הצעדים x) | הפעולה מזיזה את הצב x צעדים קדימה.  הערה: אם זנבו של הצב מופנה כלפי מטה, הצב ישאיר אחריו קו. |
| tailDown() | הפעולה מורידה את זנב הצב. |
| tailUp() | הפעולה מרימה את זנב הצב. |
| turnLeft(מספר הצעדים x) | הפעולה מפנה את פני הצב x מעלות נגד כיוון שעון. |
| turnRight(מספר הצעדים x) | הפעולה מפנה את פני הצב x מעלות עם כיוון שעון. |
| clear() | הפעולה מוחקת את כל הקווים שהצב השאיר אחריו. |

##### סיכום פרק המבוא

* **אלגוריתם** הוא רישום ראשוני (סקיצה) לפתרון בעיה במספר צעדים סופי.
* **בבדיקת** אלגוריתם צריך לבדוק שלושה תנאים:

פעולות האלגוריתם מבוצעות לפי סדר הופעתן כל עוד לא נאמר אחרת.

כאשר ביצוע פעולה מסתיים, ממשיכים לפעולה הבאה עד שנגמרות הפעולות.

לרוב נעדיף שביצוע האלגוריתם יסתיים אחרי מספר סופי של פעולות.

* **בעיה אלגוריתמית** – בעיה שעבורה מוגדרות מטרה סופית ונקודת מוצא שממנה מתחילים.
* עבור בעיה נתונה אפשר להציג **מספר פתרונות**, כלומר אלגוריתמים שונים.
* **יעילות** של אלגוריתם תלויה במספר הפעולות שהוא מבצע.
* **נכונות** של אלגוריתם נבחנת בהשגת המטרה ובכך שלא נעשו שגיאות להשגתה במהלך הדרך .
* **עצמים בעולמנו**: כל פריט בעולם – חי, צומח, דומם, מופשט – הוא עצם שאפשר לאפיין באמצעות תכונותיו, ובדרך כלל גם באמצעות פעולותיו.
* כאשר אנו רוצים להשתמש **בעצם** עלינו ליצוראחד כזה. יצירת עצם נעשית באמצעות הפעולה**new** , לדוגמה:

Organ myOrgan = new Organ() ;

* כאשר אנו רוצים להורות לעצם לבצע פעולה עלינו לרשום את שם העצם ואחריו נקודה **.** לאחר הנקודה נרשום את שם הפעולה שהעצם מתבקש לבצע, לדוגמה:

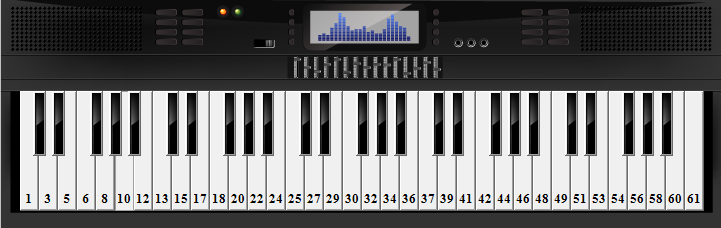
myOrgan.play(3);

##### חמישה בחמש

לפניכם רשימת נושאים לעבודת חקר קצרה.  
**בחרו** את אחד מהנושאים ברשימה, ו**הכינו** מצגת בת חמישה שקפים.  
לרשותכם חמש דקות להציג בפני הכיתה את הנושא הנבחר.

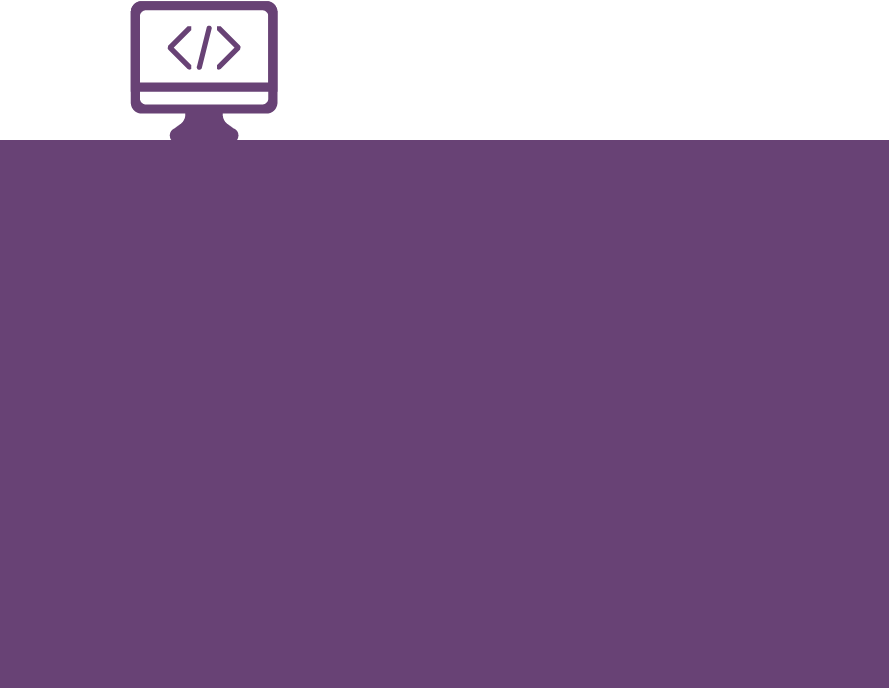
1. מדוע אין פרס נובל במקצוע מדעי המחשב? איזה פיצוי מקבלים חוקרים מצטיינים במדעי המחשב?
2. איזו שפת תכנות הוצגה בתאריך 10.12.1980, ומדוע הוכרז שמה דווקא בתאריך זה?
3. מה הקשר של המושג **BUG** (באג) למדעי המחשב? (הכוונה **אינה** לרשת החנויות הקרויה בשם זה)
4. מי אמר את המשפט: "במתמטיקה אתה לא מבין דברים, אתה רק מתרגל אליהם"? מה הקשר שלו להתפתחות מדעי המחשב?
5. הצצה לתוך המחשב: הסבירו באמצעות תמונות, סרטונים או הדגמה חיה את מבנה המחשב.
6. כיצד מכונת אריגה הביאה לפריצת דרך במדעי המחשב?
7. הפעם הראשונה שנעשה שימוש במחשב היה בשנת 1973 בשדרה השישית בלב מנהטן. מה קרה לו מאז ועד היום?
8. כל מה שאנו רוצים לדעת ולא העזנו לשאול ניתן למצוא בעולם המופלא המכונה WWW . האם זו טענה נכונה?
9. מהו פיאנולה (pianola) ומה הקשר שלו למדעי המחשב?

משימה 9: האורגן שלנו



## שער 2:

מדעי המחשב



פרק 2: הכרות עם מושגי יסוד בתכנות  
*הקדמה*

בפרק הקודם הכרנו חלק ממגוון התחומים שבהם עוסק עולם מדעי המחשב, ובהם את שתי אבני היסוד **הבעיה האלגוריתמית** ו**האלגוריתם כפתרון לבעיה**. ככל שנתקדם בספר זה הבעיות ייעשו מורכבות יותר ויותר, ולעיתים ידרשו מכם "קפיצות מחשבה" כדי להתאמן בשיטות שונות של פתרון בעיות.

רגע חושבים!

אילו מכשירים מתוחכמים אתם מכירים שמשתמשים באלגוריתמים?



##### משימה 1: מיקרוסקופ תנועה

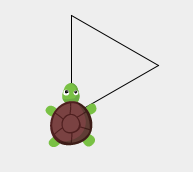
1. **צפו** בהרצאת TED של מיכאל רובינשטיין, [לראות תנועה נסתרת, לשמוע קולות נעלמים](https://www.ted.com/speakers/michael_rubinstein).
2. **השיבו** על השאלות הבאות:
3. איזה סרטון הייתם מצלמים ומנתחים באמצעות מיקרוסקופ התנועה?
4. מה בעצם "רואה" המיקרוסקופ?
5. בסרטון נאמר שהמיקרוסקופ משתמש בעיבוד תמונה. מה לדעתכם המשמעות של מונח זה? כתבו במילים שלכם.

כפי שלמדנו מתרגיל זה, יש רמות עומק שונות שבהן ניתן לעסוק במדעי המחשב. בפרק זה נעסוק במספר מושגים שירחיבו את יכולותיכם בפתרון בעיות במדעי המחשב. עדיין לא נעסוק בעיבוד תמונה אבל נפתור בעיות אחרות הדורשות שימוש באלגוריתמים מורכבים יותר, ועבורן יידרשו כלים חדשים. בהרצאה שמעתם על ניתוח בהירות הפיקסלים לאורך זמן והגברת השינויים, והמושגים הללו נשמעים מורכבים, אבל הם יהיו מובנים יותר בהמשך לימודיכם ותוכלו להבין איך אפשר להשוות בין ערך חדש לערך קיים עבור כל פיקסל בתמונה.

##### משתנים וטיפוסי נתונים

רגע חושבים!

באילו שיטות ואמצעים אנחנו שומרים על דברים בעולם?

כיצד יכול אלגוריתם להתייחס למידע שנוצר בשלבים הקודמים? הוא ירשום לעצמו או ישמור את המידע במקום שאליו יוכל לגשת בזמן הרצוי.

ברצוננו לכתוב תוכנית אשר תצייר בעזרת הצב משולש שווה צלעות. אורך הצלע ייקבע על ידי המשתמש.

לצורך כך נכתוב הודעה למשתמש ונבקש ממנו להקליד את המספר הרצוי לו עבור אורך הצלע, ואת המספר הזה נצטרך לשמור כדי שנוכל לקדם את הצב 3 פעמים לפי אורך זה.

מעיון בטבלת הממשק למשתמש, נתן להסיק שהצב אינו יודע לשמור ערכים מספריים. לכן נצטרך לשמור בעצמנו את הנתון שנקלט בתוכנית שלנו. יש לשמור את הנתונים כך שניתן יהיה לאחזר (לשלוף) אותם כאשר נרצה, וכן נוכל לבצע בעזרתם פעולות שונות כמו הוספת ערך, שינוי ועוד.

אמצעי שמירת המידע הראשון שנכיר נקרא **משתנה**.

**משתנה** (variable) או **תכונה** (property) הוא מקום בזיכרון של המחשב שבמהלך ביצוע האלגוריתם אפשר לשמור בו ערך המייצג תכונה, למשל גובה, צבע, תווית, קיבולת וכדומה. המשתנה מאפשר לקרוא את הערך השמור בו ולעדכן אותו בשעת הצורך. המידע השמור במשתנה נקרא ערך המשתנה, וכדי להשתמש בו יש לתת למשתנה שם.

כך נכתוב הוראה להגדרת משתנה בשם number המכיל מספר שלם:

int number;

הגדרה זו כוללת שני חלקים. החלק הראשון הוא הגדרת הטיפוס – int, שהוא קיצור של המילה האנגלית integer שמשמעה **מספר שלם**. החלק השני הוא שם המשתנה שאליו נוכל לפנות, במקרה הזה number.

במשתנים ניתן לרשום בכל פעם רק ערך אחד, אבל כמו את השם, אפשר לשנות אותו. נראה דוגמאות נוספות להוראות עם מספרים שלמים:

number = 0;

פעולה זו, שבה רושמים ערך למשתנה, נקראת **השמה** או **הצבה** והיא מתבצעת באמצעות אופרטור ההשמה המסומן בסימן **=**.

אפשר לשלב הגדרה והשמה בהוראה אחת השקולה לשתי ההוראות:

int myNumber = 1;

כמו כן, אפשר להגדיר כמה משתנים באותה הגדרה עם השמה או בלעדיה:

int side= 80 , angle=90, number;

לפניכם הזדמנות ליישם את מה שלמדתם:

### 

### משימה 2: מציירים משולש

1. **הגדירו** משתנה מטיפוס שלם בשם side שייצג את אורך הצלע הרצוי. השימו בו את הערך 100.
2. **הגדירו** משתנה מטיפוס שלם בשם angle שייצג את הזווית שאליה פונה הצב. השימו בו את הערך 60.
3. ציירו משולש שווה צלעות שאורך כל צלע הוא side צעדים, קודקודו השמאלי התחתון נמצא במיקומו הראשוני של הצב, ואחת מצלעותיו אנכית.

**פתרון:**

1. int side = 100;
2. int angle = 60;
3. ציור המשולש:

**int** side = 100;

**int** angle = 120;

Turtle leonardo = **new** Turtle();

leonardo.tailDown();

leonardo.moveForward(side);

leonardo.turnRight(angle);

leonardo.moveForward(side);

leonardo.turnRight(angle);

leonardo.moveForward(side);

leonardo.turnRight(angle);

רגע חושבים!

בכמה משתנים הייתם משתמשים כדי לצייר משולש ישר-זווית ושווה-שוקיים?

כזכור, מקומות שמירת המידע נקראים משתנים משום שאפשר לשנות את הערך השמור בהם.

ההוראה angle = 90;תשנה את ערך המשתנה angle ל-90 מעלות, והערך שהיה רשום לפני השינוי יימחק.

לכן, במקרה של ציור משולש ישר-זווית ושווה-שוקיים, אפשר להתחיל את התוכנית עם זווית של 45מעלות ולשנות את ערך המשתנה במהלך התוכנית.



##### משימה 3: משולש ישר-זווית

1. **כתבו** אלגוריתם לציור משולש ישר-זווית עם הזוויות 90, 60 ו-30.   
   ניתן לצייר את המשולש בכל אופן שתרצו – כשהוא עומד על קודקודו או כשהוא עומד על אחת מצלעותיו.
2. **ממשו** את האלגוריתם בעזרת קטע תוכנית.

##### קלט/פלט

רגע חושבים!

המטרה שלנו הייתה לצייר בעזרת הצב משולש שווה-צלעות שהצלע שלו היא בגודל שבוחר המשתמש.

אנחנו מחפשים שיטה להעביר נתונים לתוכנית, כך שנתון שיזין המשתמש יישמר במשתנה והצב יוכל לעשות בו שימוש. העברת הנתונים מהמשתמש אל התוכנית בשלב זה תיעשה באמצעות המקלדת. כדי לבקש נתונים מהמשתמש יש לרשום עבורו הודעה שתבהיר לו אילו נתונים נדרשים ממנו.

בסביבת העבודה שלנו האזור שבו אפשר להציג למשתמש הודעות ולקבל ממנו נתונים נקרא Console, כלומר צג המחשב במקרה זה.

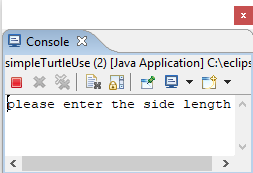
### 

### משימה 4: קולטים נתונים מהמשתמש

1. **הדפיסו** הודעה למשתמש שבה אתם מבקשים ממנו להקליד את אורך הצלע הרצוי.
2. **הגדירו** משתנה מטיפוס שלם בשם side שייצג את אורך הצלע וקלטו אליו את האורך הרצוי מהמשתמש.
3. **ציירו** משולש שווה-צלעות שאורך כל צלע בו הוא side צעדים, קודקודו השמאלי התחתון נמצא במיקומו הראשוני של הצב, ואחת מצלעותיו אנכית.

**הדפסה**, שנקראת גם פלט, משתמשת בפעולות מוכנות בסביבת העבודה. בשפת java הפלט יוצג בחלונית שנקראת console. היישום של הוראת הפלט בשפת java ייראה כך:

System.out.println(“Please enter the side length”);

ביישום הוראה זו יודפס המשפט בחלונית. מיקום החלונית משתנה בהתאם להגדרות סביבת העבודה.

השימוש בפלט מאפשר לכותבי התוכנית להציג הודעות למשתמש, לאחר שהוצגה ההודעה נרצה לקלוט את תשובתו. לכן התוכנית צריכה להיות ערוכה לקליטת הנתון שיוקלד.

ההיערכות כוללת שלושה שלבים:

הראשון הוא ייבוא של מחלקה (ספרייה) המאפשרת להגדיר אובייקט שבעזרתו נוכל לקלוט נתונים. כך לדוגמה נעשה בייבוא מחלקת הצב כדי ליצור אובייקט שיוכל לצייר על המסך. הוראת הייבוא תראה כך:

import java.util.Scanner;

השלב השני הוא הגדרת אובייקט מסוג Scanner באמצעות ההוראה הבאה:

Scanner myScanner = new Scanner (System.in);

המילה myScanner היא שם האובייקט שהוגדר. כמו שהגדרנו צב בשם leonardo, גם כאן אפשר להשתמש בכל שם שנרצה. רצוי לבחור שם משמעותי כמו myScanner או reader שמרמז על תפקיד העצם שהוגדר.

השלב השלישי יהיה להכין משתנה מתאים (הפעם מטיפוס שלם):

int side;

ואז להשתמש בהוראה שקולטת את המספר שרשם המשתמש בעזרת האובייקט myScanner שיצרנו:

side = myScanner.nextInt();

ביצוע הוראה זו יעכב את המשך ביצוע התוכנית עד שהמשתמש יקליד מספר בחלונית ה-console וילחץ על המקש Enter. לאחר הלחיצה על המקש, המספר ייקלט ויישמר במשתנה side.

**הוראה זו נקראת הוראת קלט, והערך שנקלט נקרא קלט.**

רגע חושבים!

מה קורה אם המשתמש השובב שלנו יזין מספר שאיננו שלם?

זאת שאלה מעולה שאליה נחזור בהמשך הפרק. בשלב זה נניח שהמשתמש שלנו ממושמע ומקליד רק מספרים שלמים.

לפניכם התוכנית המלאה, כולל המחלקות הנדרשות לייבוא ושאר ההגדרות. אך בתוכנית חסרה עדיין הזווית שבה הצב צריך לפנות. חשבו מה הזווית הנכונה, השלימו ובדקו.

import java.util.Scanner;

import unit4.turtleLib.Turtle;

public class SimpleTurtleUse {

public static void main(String[] args) {

Scanner myScanner = new Scanner(System.in);

int side = 0;

int angle = \_\_\_\_\_;

System.out.println("Please enter the length of the side of the triangle you want");

side = myScanner.nextInt();

Turtle leonardo = new Turtle();

leonardo.tailDown();

leonardo.moveForward(side);

leonardo.turnRight(angle);

leonardo.moveForward(side);

leonardo.turnRight(angle);

leonardo.moveForward(side);

leonardo.turnRight(angle);

}

}

C#

using System;

using Unit4.TurtleLib;

class SimpleTurtleUse {

static void Main(string[] args) {

int side = 0;

int angle = \_\_\_\_\_;

Console.WriteLine("Please enter the length of the side of the triangle you want");

side = int.Parse(Console.ReadLine());

Turtle leonardo = new Turtle();

leonardo.TailDown();

leonardo.MoveForward(side);

leonardo.TurnRight(angle);

leonardo.MoveForward(side);

leonardo.TurnRight(angle);

leonardo.MoveForward(side);

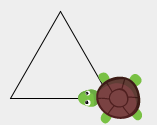
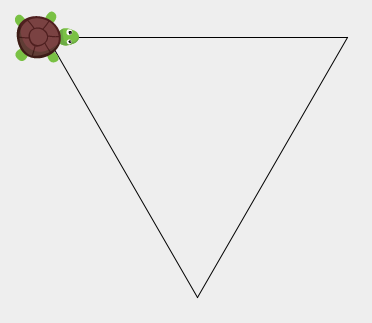
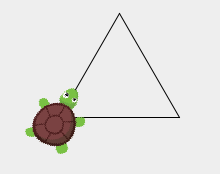
leonardo.TurnRight(angle);

}

}



##### משימה 5: הצב מצייר לפי הגדרת אורך הצלע וזווית התחלה מבוקשת



אורך צלע 120, זווית התחלה 30 אורך צלע 300, זווית התחלה 90 אורך צלע 100, זווית התחלה 270

1. **פתחו** אלגוריתם אשר בונה משולש לפי אורך צלע וזווית התחלה שמזין המשתמש.

**קלט:** אורך צלע המשולש וזווית התחלה רצויה.

**פלט:** ציור של המשולש בהתאם לקלט

**שימו לב**! התייחסו לכלל המקרים האפשריים, ולא רק לדוגמאות המצוירות מעלה.

1. **ממשו** את האלגוריתם בתוכנית מחשב.

ניתן להיעזר בשלבים הבאים:

1. **הדפיסו** הודעה למשתמש שבה אתם מבקשים ממנו להקליד את אורך הצלע הרצוי.
2. **קלטו** למשתנה side נתון שייצג את אורך הצלע הרצוי שהקליד המשתמש.
3. **הדפיסו** הודעה למשתמש שבה אתם מבקשים ממנו להקליד את הזווית שבה אתם רוצים שהצלע הראשונה של המשולש תיטה.
4. **קלטו** למשתנה firstAngle את הזווית הרצויה שהקליד המשתמש.
5. **ציירו** משולש שווה-צלעות שאורך כל צלע בו הוא side צעדים, קודקודו השמאלי התחתון נמצא במיקומו הראשוני של הצב, ואחת מצלעותיו נוטה בזווית firstAngle

### משתנה ממשי

אחד התנאים להצלחת התוכנית שהוצגה היה כאמור ההנחה שהמשתמש מקליד מספר שלם. אולם מה אם ברצונו לצייר משולש שאורך הצלע שלו היא מספר לא שלם של צעדים? איך נקלוט את המספר 70.5? הוראת הקלט myScanner.nextInt(); היא בהכרח לא מתאימה, שכן היא מסתיימת ברצף התווים int שמבהירים כי היא קולטת מספרים שלמים בלבד. לכן אנו זקוקים לטיפוס נתונים חדש, טיפוס נתונים למספרים ממשיים. טיפוס נתונים זה נקרא double.

הגדרת משתנה מטיפוס ממשי תיעשה באמצעות ההוראה:

double side;

בהתאם, קלט למשתנה ממשי ייעשה באמצעות ההוראה:

side = myScanner.nextDouble();

### 

### משימה 6: קולטים נתונים ממשיים

**שנו** את תוכנית שרטוט המשולש שווה-הצלעות כך שניתן יהיה לצייר משולשים בעלי צלעות שאורכן אינו מספר צעדים שלם. כמו כן **אפשרו** זווית הטיה של מספר לא שלם לזווית הצלע הראשונה שמצוירת.

**הפתרון:**

import java.util.Scanner;

import unit4.turtleLib.Turtle;

public class DoubleInput {

public static void main(String[] args) {

Scanner myScanner = new Scanner(System.in);

double side = 0;

double firstAngle = 0;

System.out.println("please enter the length of

the side of the triangle you want");

side = myScanner.nextDouble();

System.out.println("please enter the angle to

start drawing from");

firstAngle = myScanner.nextDouble();

Turtle leonardo = new Turtle();

leonardo.tailDown();

leonardo.turnRight(firstAngle);

leonardo.moveForward(side);

leonardo.turnRight(120);

leonardo.moveForward(side);

leonardo.turnRight(120);

leonardo.moveForward(side);

leonardo.turnRight(120);

}

}

C#

using System;

using Unit4.TurtleLib;

class DoubleInput {

static void Main(string[] args) {

double side = 0;

double firstAngle = 0;

Console.WriteLine("please enter the length of   
 the side of the triangle you want");

side = double.Parse(Console.ReadLine());

Console.WriteLine("please enter the angle to   
 start drawing from");

firstAngle = double.Parse(Console.ReadLine());

Turtle leonardo = new Turtle();

leonardo.TailDown();

leonardo.TurnRight(firstAngle);

leonardo.MoveForward(side);

leonardo.TurnRight(120);

leonardo.MoveForward(side);

leonardo.TurnRight(120);

leonardo.MoveForward(side);

leonardo.TurnRight(120);

}

}

##### ביטויים חשבוניים

בשלב הזה אנו יודעים איך להגדיר משתנה. אנו יודעים כיצד לשים בו ערך וראינו שערך המשתנה ניתן להחלפה.

רגע חושבים!

מה הכוונה במונח "ביטוי חשבוני"? אילו פעולות חשבוניות ניתן לעשות עם מספרים השמורים במשתנים?

לפניכם משחק קצר המדגים מה הכוונה במונח "ביטוי":

ראינו סוגי ביטויים שונים, ובהם גם את הביטוי החשבוני. אנו מכירים פעולות מתמטיות שונות שניתן לבצע על מספרים, למשל לחבר ביניהם או להכפיל אותם. פעולות חשבוניות אלו נקראות גם **אופרטורים**. התוצאה של פעולת האופרטור נקבעת על פי טיפוסי הנתונים שעליהם הוא פועל, והם קראים האופרנדים שלו.

ביטוי אשר מורכב מ**אפרטורים חשבוניים** בין ערכים ו/או משתנים מטיפוס ממשי או שלם נקרא **ביטוי חשבוני**.

התוצאה של ביטוי חשבוני תהיה בעלת ערך מספרי מטיפוס שלם או ממשי, בהתאם לפעולות המתבצעות בו ולערכים שבו. אם כל המרכיבים של ביטוי חשבוני הם מטיפוס שלם, אזי גם הטיפוס של הביטוי החשבוני יהיה שלם. אם נכלל בביטוי ערך מטיפוס ממשי, או שתוצאת אחת הפעולות עשויה להיות מספר לא שלם, טיפוס הביטוי החשבוני יהיה ממשי.

למשל, הערך של הביטוי 5 + 8 יהיה 13, והוא יהיה מטיפוס שלם. אך הערך של הביטוי 5.0 + 8 יהיה 13.0, והוא יהיה מטיפוס ממשי. בהמשך נרחיב על טיפוסי ביטויים חשבוניים.

בביטוי חשבוני מתקיים סדר פעולות החשבון המוכר לנו: לסוגריים העדיפות הגבוהה ביותר, לכפל ולחילוק עדיפות נמוכה יותר, ולחיבור ולחיסור העדיפות הנמוכה ביותר.

ביטויים חשבוניים מאפשרים כאמור שילוב של ערכים השמורים במשתנים, כך שאם הוגדר המשתנה int price = 100; , הביטוי החשבוני price + 21 ניתן לחישוב וערכו 121.

ניתן לשמור את תוצאת החישוב של ביטוי חשבוני במשתנה מהטיפוס המתאים על ידי הוראת השמה:

int newPrice = price + 21;

חשוב לציין שהשמה כגון זו מבוצעת במספר שלבים: ראשית מחושבת תוצאת הביטוי החשבוני שמימין לאופרטור ההשמה, ולאחר מכן נבדקת התאמת הטיפוס של תוצאת הביטוי לטיפוס המשתנה שמשמאל לאופרטור ההשמה. רק אם נמצאה התאמה מתבצעת השמה למשתנה.

נראה מספר מקרים מייצגים:

ההוראה:

double result1 = 4.3 + 5.7;

היא הוראה תקינה, שכן תוצאת החישוב היא מספר מטיפוס ממשי, וגם המשתנה שבו אנו מבקשים לשמור אותה הוא מטיפוס ממשי. לכן ההשמה תתבצע.

ההוראה:

int result2 = 4.3 + 5.7;

**היא הוראה לא תקינה**, שכן תוצאת החישוב היא מספר מטיפוס ממשי, אך המשתנה שבו אנו מבקשים לשמור אותה הוא מטיפוס שלם בלבד. לכן ההשמה לא תתבצע.

לעובדה ש**ערך** הביטוי הוא 10.0, כלומר מספר שלם, אין משמעות בבדיקת התקינות של ההשמה שכן הטיפוס אינו מתאים.

בשונה מהדוגמה האחרונה, אם תוצאת חישוב הביטוי החשבוני תהיה מספר שלם וננסה לבצע השמה שלו למשתנה מטיפוס ממשי, ההשמה תתאפשר משום שהערך השלם יומר אוטומטית לערך ממשי שקול.

double result = 3 + 4;

היא הוראה חוקית, והערך שיירשם במשתנה result יהיה 7.0. אנו כותבים 7.0 כדי להדגיש שזהו ערך מטיפוס ממשי, אך ההמרה האוטומטית מתבצעת באופן אוטומטי, כיוון שקבוצת המספרים השלמים היא תת-קבוצה של המספרים הממשיים: הערך 7 זהה לערך 7.0. לכן לא נאבד מידע בפעולת ההמרה.

##### משימה 7: הגדרה והשמה עם אופרטורים

לפניכם מספר הוראות הגדרה והשמה, העלו השערות איזו מהן חוקית ואיזו לא, ובדקו במחשב.

1. double steps = 10.5 + 3;
2. int moreSteps = steps - 0.5;
3. int days = 7.0;
4. double nights = 7;
5. double interestinOption1 = 1.5 \* 1.5;
6. int interestinOption2 = 1.5 + 1.5;

הפעולות החשבוניות שהוצגו עד כה, כגון חיבור, חיסור וכפל, יתנו תוצאה שלמה אם יפעלו על מספרים שלמים. לעומתן, פעולת החילוק בין מספרים שלמים יכולה לתת תוצאה שאינה מספר שלם. בפעולה זו נדון בהמשך.

רגע חושבים!

עם הכלים שלמדתם עד כה, כיצד ניתן להדפיס במסך הפלט את סך הצעדים שעבר הצב?

לצורך כך אנחנו זקוקים למשתנה שבעזרתו נצבור את כל הצעדים של הצב. כדי לעשות זאת נצטרך להוסיף את הערך החדש לערכו של משתנה הסכום, ואת התוצאה להשים במשתנה הסכום.

למשל, אם במשתנה length נשמר אורך המסלול עד כה, אז הוספה של 40 צעדים תיבנה כך:

length = length + 40;

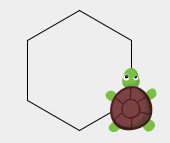
גם כאן, קודם מתבצע חישוב הביטוי החשבוני שמימין לאופרטור ההשמה, ולאחר מכן מתבצעת בדיקת התאמת הטיפוס המחושב לטיפוס המשתנה שבו אמור להיות מושם הערך, ואז מתבצעת השמת התוצאה למשתנה.

אם מספר הצעדים בכל תנועה שמור במשתנה אחר, למשל side, ההוספה תתבצע באותה צורה:

length = length + side;



##### משימה 8: סך הצעדים של הצב

צלע בגודל 60 צלע בגודל 20

**היעזרו** בתוכנית שרטוט המשולש שווה הצלעות כדי לפתור את המשימה הבאה:

**קלט**: מספר שלם שמייצג את אורך הצלע במשושה שווה-הצלעות.

**פלט**: משושה שווה-צלעות שאורך כל צלע בו נקבע על בסיס הקלט מהמשתמש. הצלע הראשונה שתצויר תהיה אנכית, ונקודת ההתחלה שלה תהיה במיקום ההתחלתי של הצב. המשושה יצויר משמאל לצלע הראשונה.

בסיום הציור תודפס הודעה המציגה את המרחק שעבר הצב במהלך השרטוט.

##### חלוקת שלמים

נרחיב כעת על פעולת החלוקה בין שני מספרים שלמים. זוהי פעולה מיוחדת שזכתה לשם מיוחד – **חלוקת שלמים**.

בחיינו יש מקרים רבים שבהם נעסוק במספרים שלמים בלבד: למשל אם לטיול השנתי יצאו 60 תלמידים, ובכל אוטובוס ניתן להסיע 40 תלמידים בלבד, לא נוכל להזמין אוטובוס וחצי (60/40=1.5). אם ניתן לקופאית בסופר שקית עם כל המטבעות שאספנו בבית ונבקש ממנה לתת לנו את הסכום בשטרות של 20 שקלים, והיא תספור את המטבעות ותגלה שיש לנו 90 שקלים, היא לא תוכל לתת לנו ארבעה וחצי שטרות (90/20=4.5).

אך אופרטור החילוק, המיוצג על ידי הסימן ( / ), יפעל בצורה שונה בהתאם לטיפוס המשתנים שעליהם הוא פועל: בפעולה על ערכים שלמים יוחזר ערך מטיפוס שלם, ובפעולה על ערכים ממשיים יוחזר ערך מטיפוס ממשי.

דוגמאות: פעולת חילוק בערכים מטיפוס שלם:

* המנה של 13/2 היא 6.
* המנה של 96/10 היא 9.

בהתאמה, ניתן להשים ביטוי חשבוני של חלוקת שלמים.

חישוב והשמה לדוגמה:

int answer = 19 / 5;

ההוראה תחשב את הביטוי החשבוני 19/5. בגלל ששני המספרים הם שלמים, תתבצע פעולה של חלוקת שלמים, וערך הביטוי יהיה 3. ההוראה תגדיר משתנה מטיפוס שלם בשם answer, ומכיוון שהטיפוסים מתאימים היא תשמור בו את הערך 3.

חישוב והשמה נוספת לדוגמה:

double answer = 19 / 5;

ההוראה תחשב את הביטוי החשבוני 19/5, ובגלל ששני המספרים הם מספרם שלמים, תתבצע פעולה של חלוקת שלמים וערך הביטוי יהיה 3. ההוראה תגדיר משתנה מטיפוס ממשי בשם answer ותבדוק את התאמת הטיפוסים לפני ההשמה, ולמרות שאין ביניהם תתבצע המרה אוטומטית – המתבצעת במקרה של שמירת ערך מטיפוס שלם במשתנה מטיפוס ממשי. בסיום הפעולה, הערך שיירשם במשתנה יהיה 3.0



משימה 9: מה משמעות הוספת נקודה עשרונית למספר

**בדקו במחשב** מה יהיה הפלט עבור ההוראות הבאות **ונסו** לגבש תשובה לשאלה.

1. System.out.println(5 / 4);

2. double result = 5 / 4;

3. int answer = 5 / 4;

4. System.out.println(5.0 / 4.0);

5. System.out.println(5.0 / 4);

6. System.out.println(5 / 4.0);

C#

1. Console.WriteLine(5 / 4);

2. double result = 5 / 4;

3. int answer = 5 / 4;

4. Console.WriteLine(5.0 / 4.0);

5. Console.WriteLine(5.0 / 4);

6. Console.WriteLine(5 / 4.0);

בשלוש האפשרויות הראשונות, המחלק והמחולק הם מספרים שלמים ולכן תתבצע חלוקת שלמים ותוצאת הביטוי החשבוני תהיה שלמה גם כן. הערך שמתקבל יהיה בשלושת המקרים 1.

בשלוש האפשרויות הבאות, לפחות אחד מהמספרים כולל נקודה עשרונית ותוצאת החישוב היא 1.25. במקרה לא מתבצעת חלוקת שלמים אלא חלוקה עשרונית, ומכאן שמספר עם הנקודה העשרונית נחשב ממשי.

נמשיך ונחקור את פעולתו המותאמת של אופרטור החילוק, הנקבעת על פי טיפוסי הנתונים או המשתנים שעליהם הוא פועל. בדקו במחשב מה יקרה כשנשתמש במשתנים:

int number1 = 9;

int number2 = 4;

System.out.println(number1/number2);  
double number3 = 9;

int number4 = 4;  
System.out.println(number3/number4);

int number5 = 9;

int number6 = 4;

double number7 = number5/number6;

System.out.println(number7);

C#

int number1 = 9;

int number2 = 4;

Console.WriteLine(number1/number2);  
double number3 = 9;

int number4 = 4;  
Console.WriteLine(number3/number4);

int number5 = 9;

int number6 = 4;

double number7 = number5/number6;

Console.WriteLine(number7);

### 

### משימה 10: חלוקת סוכריות בכיתה

לפניכם שאלה לחישוב שארית בחלוקת שלמים.

תלמיד החליט לחגוג את יום הולדתו בכיתה ולהביא סוכריות לחברי כיתתו.

הוא ביקש את אישור המורה לכך, והיא התנתה את אישורה בכך שכל תלמיד ותלמידה יקבלו את אותו מספר סוכריות, ושהיא תדע מראש כמה סוכריות יקבל כל אחד כדי לא לפגוע במהלך השיעור.

התלמיד החליט לחלק את הסוכריות שווה בשווה בין כל התלמידים כך שכל תלמיד ותלמידה יקבלו את המספר הגדול ביותר של סוכריות האפשרי בחלוקה שווה לכולם.

הוא החליט לכתוב תוכנית מחשב שתקל עליו לפתור את בעיית חלוקת הסוכריות.

**קלט**: מספר הסוכריות לחלוקה ומספר התלמידים בכיתה.

**פלט**: מספר הסוכריות שיקבלו כל תלמיד ותלמידה ומספר הסוכריות הנותרות.

**פתחו** אלגוריתם הפותר את בעיית חלוקת הסוכריות.

**ממשו** את האלגוריתם באמצעות תוכנית מחשב שתקלוט מהמשתמש את נתוני הקלט ותדפיס את נתוני הפלט.

ניתן את דעתנו לעניין מספר הסוכריות שיישארו אצל התלמיד מבלי שיוכל לחולק אותן לאחרים. נשים לב שמספר הסוכריות שיישארו לתלמיד חייב להיות קטן ממספר תלמידי הכיתה, אחרת אפשר היה להוסיף סוכריה אחת לפחות לכל תלמיד.

ניתן לחשב מספר זה בחיסור סך כל הסוכריות שיחולקו ממספר הסוכריות לחלוקה.

למשל אם יש לו 405 סוכריות ו-37 תלמידים בכיתה, אז כל תלמיד ותלמידה יקבלו 405/39=10, סוכריות, ואת מספר הסוכריות שיישאר בידי המחלק אפשר לחשב בעזרת הביטוי: 405 – 10\*39 = 26. כלומר, כל תלמיד ותלמידה יקבלו 10 סוכריות, בסך הכול יחולקו 390 סוכריות, ובידי התלמיד המחלק יישארו בסוף החלוקה 26 סוכריות.

ישנו סימן חלוקה מיוחד אשר מבצע את פעולת השארית מבלי שיהיה צורך לבצע את החלוקה המקורית. זהי פעולה מתמטית (כלומר **אופרטור** מתמטי) אשר סימנה בשפת המחשב הוא **%** (אין קשר לאחוזים). ההוראה:

int remainder= 234 % 50;

נרשום את הערך 34 במשתנה remainder (בחירת השם היא חופשית, בחרנו בשם זה כי remainder באנגלית היא שארית). תוכנית המחשב תראה כך:

import java.util.Scanner;

public class DivAndModeChapter2 {

public static void main(String[] args) {

Scanner reader = new Scanner(System.in);

System.out.println("Please enter number of

candies");

int numberOfCandies = reader.nextInt();

System.out.println("Please enter number of

students");

int numberOfStudents = reader.nextInt();

int candiesForStudent = numberOfCandies / numberOfStudents;

System.out.print("Each student will get: ");

System.out.print(candiesForStudent);

System.out.println(" candies.");

int candiesLeft = numberOfCandies % numberOfStudents);

System.out.print("You will be left with: ");

System.out.print(candiesLeft);

System.out.println(" candies.");

}

}

C#

using System;

class DivAndModeChapter2 {

static void Main(string[] args) {

Console.WriteLine("Please enter number of

candies");

int numberOfCandies =   
int.Parse(Console.ReadLine());

Console.WriteLine("Please enter number of

students");

int numberOfStudents =

int.Parse(Console.ReadLine());

int candiesForStudent = numberOfCandies /

numberOfStudents;

Console.WriteLine("Each student will get: ");

Console.WriteLine(candiesForStudent);

Console.WriteLineln(" candies.");

int candiesLeft = numberOfCandies %

numberOfStudents);

Console.WriteLine("You will be left with: ");

Console.WriteLine(candiesLeft);

Console.WriteLineln(" candies.");

Console.ReadKey();

}

}

טיפוס הנתונים **מחרוזת** (String)

בתוכניות שכתבנו עד כה השתמשנו בשני טיפוסי נתונים שמוגדרים בשפה: הטיפוס הממשי double והטיפוס השלם int. למעשה פגשנו סוג נוסף של נתונים, אבל לא התייחסנו אליו כטיפוס נתונים שכן ערכו לא נשמר במשתנה: כאשר הדפסנו הודעות למשתמש ורשמנו תווים (אותיות או מספרים) בין גרשיים, יצרנו **מחרוזת** ושלחנו אותה להדפסת פלט. כך למשל בהוראה:

System.out.println("Please enter number of candies");

Console.WriteLineln("Please enter number of candies");

השתמשנו במחרוזת "Please enter number of candies". המחרוזת היא טיפוס נתונים שונה מהמשתנים הפשוטים שהכרנו עד כה, והיא למעשה עצם בדומה לצב. בשפת JAVA מוגדרת המחלקה מחרוזת String, ולכן אין צורך לייבא אותה כמו שעשינו למחלקת הצב, אך הגדרת עצם מסוג מחרוזת מתבצעת בדומה להגדרת הצב.

String name = new String(“Avraham”);

string name="Avraham";

בהוראה זו יצרנו עצם חדש בעל תכונה אחת, והיא ערכו – רצף התווים Avraham. המחלקה String מאפשרת הדפסה ישירה של העצם בצורה הבאה:

System.out.println(name);

Console.WriteLineln(name);

השימוש במחרוזות נפוץ במיוחד, ולכן הוגדרה אפשרות להגדיר משתנה מסוג מחרוזת ולאתחל אותו מבלי להשתמש בפעולה new. זוהי כתיבה מקוצרת שקיימת לנוחות המשתמשים.

ההגדרה מתבצעת כך:

String myPetsName = “Lassie”;

שימו לב שלמרות ההגדרה "כמו של משתנה פשוט", מדובר בעצם של המחלקה String.

גם כאן ניתן לאתחל מחרוזת ריקה:

String yourName = ””;

בעבר כשפגשנו את הסימן '+' התיחסנו אליו כייצוג של הפעולה המתמטית חיבור. זהו אופרטור החיבור, וגם הוא פועל באופן שונה לפי גורמי פעולה שונים, הנקראים אופרנדים. כאשר אופרטור החיבור פועל על שני מספרים, הוא מחבר אותם בהתאם לפעולת החיבור המתמטית; ואילו כאשר הוא פועל על שתי מחרוזות, הוא ישרשר אותן למחרוזת שלישית.

את המחרוזת שנוצרה אפשר לשמור במשתנה מסוג מחרוזת. למשל, רצף ההוראות הבא יחבר או ישרשר את שתי המחרוזות ויבצע השמה של התוצאה למחרוזת שלישית

במשתנה result יושם הערך Woman Wonder. כמו כן חשוב לציין שהרווח אחרי המילה Wonder הוא תו במחרוזת, והוא נשמר בשרשור המחרוזות.

String partOne = ”Wonder“;

String partTwo = ”Woman“;

String result = partOne + partTwo;

שאלה מעניינת שנחקור היא מה קורה כאשר האופרטור + פועל על נתון מספרי ומחרוזת. ייתכן שהתשובה תפתיע אתכם.



משימה 11: היכרות עם טיפוס הנתונים מחרוזת והאופרטור +

**העריכו** מה יודפס בכל אחת מהתכניות הבאות. **בדקו** את הערכתכם.

String whatWillBePrinted = "";

whatWillBePrinted = " the answer is: 1 + 2 + 3";

System.out.println(whatWillBePrinted);

whatWillBePrinted = " the answer is: " + 1 + 2 + 3;

System.out.println(whatWillBePrinted);

whatWillBePrinted = " the answer is: " + (1 + 2 + 3);

System.out.println(whatWillBePrinted);

whatWillBePrinted = " the answer of: " + "1 + 2 + 3";

whatWillBePrinted = whatWillBePrinted + " is: " + (1 + 2 + 3);

System.out.println(whatWillBePrinted

**העריכו** מה יודפס בכל אחת מהתכניות הבאות. **בדקו** את הערכתכם.

string whatWillBePrinted = "";

whatWillBePrinted = " the answer is: 1 + 2 + 3";

Console.WriteLine(whatWillBePrinted);

whatWillBePrinted = " the answer is: " + 1 + 2 + 3;

Console.WriteLine(whatWillBePrinted);

whatWillBePrinted = " the answer is: " + (1 + 2 + 3);

Console.WriteLine(whatWillBePrinted);

whatWillBePrinted = " the answer of: " + "1 + 2 + 3";

whatWillBePrinted = whatWillBePrinted + " is: " + (1 + 2 + 3);

Console.WriteLine(whatWillBePrinted);

במשימה זו למדנו כי הפעלת האופרטורים תתבצע על פי סדר הופעתם משמאל לימין, וכן שכאשר האופרטור + פועל על מספר ומחרוזת – הוא ימיר תחילה את המספר למחרוזת, ואז ישרשר אותו אל המחרוזת שעליה פעל.

קלט של מחרוזת

### 

### משימה 12: קולטים מחרוזת

**כתבו** אלגוריתם על פי ההגדרות הבאות:

**קלט**: מחרוזת הכוללת את שמכם (למשתנה מטיפוס מחרוזת בשם name)

**פלט**: מדפיסה את ההודעה הבאה:

Good morning **הערך שנקלט**, this is your mission, should you choose to accept it.

(המשפט לקוח מסדרת הסרטים "משימה בלתי אפשרית").

**ממשו** את האלגוריתם באמצעות תוכנית מחשב.

למדנו כי לכל טיפוס הוראת קלט המתאימה לו. בדומה למספרים השלמים ולמספרים הממשיים, גם למחרוזות הוראה ייחודית.

אם למשל נגדיר אובייקט מסוג Scanner בשם reader, ההוראה תיראה כך:

String name = reader.nextLine();

1. נחלק את המשימה לשלושה חלקים:
   1. נפנה אל המשתמש בבקשה להכניס את שמו.
   2. נקלוט את השם שהוקלד ונשמור אותו במשתנה.
   3. נציג כפלט (נדפיס) הודעה משולבת מתאימה.
2. מימוש בסביבת העבודה:

System.out.println("please enter your name and press Enter");

String userName = reader.nextLine();

System.out.println("Good morning " + userName + ", this is your mission should you choose to accept it.");

C#

Console.WriteLineln("please enter your name and press Enter");

string userName = Console.ReadLine();

Console.WriteLineln("Good morning " + userName + ", this is your mission should you choose to accept

it.");

### משימה לסיכום ביניים: "מזל דלי"

לרשותכם מחלקה חדשה, bucket, שאותה תוכלו לייבא מ-unit4. המחלקה מאפשרת לראות הצגה גרפית ואנימציה של דלי שאפשר להוסיף אליו מים ולרוקן ממנו מים. אפשר גם להעביר מים מדלי לדלי. אי אפשר למלא דלי מעבר לקיבולת המקסימלית שלו.

לדלי יש שלוש תכונות:

1. הקיבולת המקסימלית שלו
2. הקיבולת הנוכחית שלו (בהתחלה היא תהיה 0)
3. תווית המספר שמופיעה עליו

לפניכם דוגמה להגדרת דלי חדש על ידי פעולה לבניית דלי בשפה חופשית:

**צור דלי1 עם התכונות (7, "דלי ראשון")**

המשמעות היא שייווצר עצם מסוג דלי שקיבולתו המקסימלית היא 7 ליטר.  
התווית שעליו תהיה **דלי ראשון**.

רגע חושבים!

מהם טיפוסי הנתונים המתאימים לכל אחת מהתכונות?

הקיבולת היא מטיפוס שלם והתווית היא מטיפוס מחרוזת.

פעולה נוספת שאפשר לבצע על הדליים היא להעביר מים מדלי לדלי.

לפניכם דוגמה להוראה בשפה חופשית של העברת מים מדלי 1 לדלי 3. ההעברה תסתיים כשדלי 3 יתמלא או כשיאזלו המים בדלי 1:

**דלי1 – שפוך מים ל(דלי 3)**



משימה 13: מזל דלי

### חלק א'

**כתבו** בשפה חופשית תוכנית המגדירה שלושה דליים:

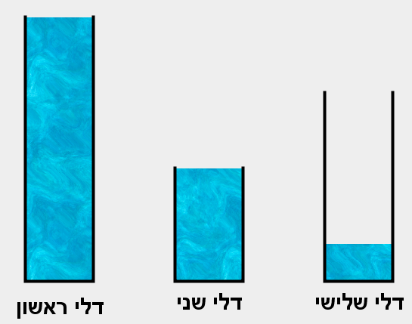
1. הראשון בקיבולת של 7 ליטרים עם התווית "דלי ראשון".
2. השני בקיבולת של 3 ליטרים עם התווית "דלי שני".
3. השלישי בקיבולת של 5 ליטרים עם התווית "דלי שלישי".

**פתרון:**

צור דלי1 עם התכונות (7, "דלי ראשון")

צור דלי2 עם התכונות (3, "דלי שני")

צור דלי3 עם התכונות (5, "דלי שלישי")

במצב ההתחלתי הדליים ריקים. ההוראות הבאות ממלאות אותם:

הפקודה "מלא דלי1 (7)" תמלא את הדלי בכמות של 7 ליטרים.

הפקודה "מלא דלי2 (7)" תמלא את הדלי בכמות של 3 ליטרים, שכן זו הקיבולת המקסימלית שלו.

הפקודה "מלא דלי3 (1)" תמלא את הדלי בכמות של ליטר אחד.

כפי שאולי הבחנתם, התוכנית לא יכולה למלא דלי מעבר לקיבולת המקסימלית שלו.

אך כאמור אפשר להעביר מים מדלי לדלי במגבלות הקיבולת, כפי שתחקרו מיד...

**הפעילו** את הסרטון הבא לצפות בהעברת מים מדלי לדלי ([קישור לסרטון](https://drive.google.com/open?id=1cqIkXcfsAnVNYzsPgjpRXBUpoTZdW9ck)).



##### משימה 14: חידת חשיבה

#### חלק א

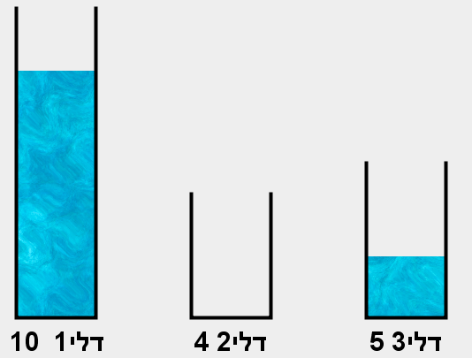
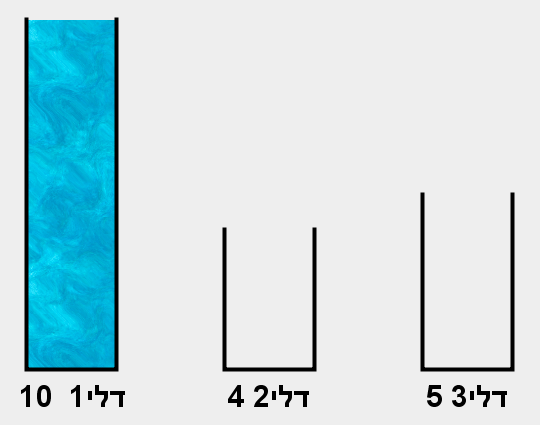
לרשותכם שלושה דליים:

* דלי ראשון בקיבולת מקסימלית של 10 ליטר, והוא **מלא** במים.
* דלי שני בקיבולת מקסימלית של 4 ליטר, והוא ריק ממים.
* דלי שלישי בקיבולת מקסימלית של 5 ליטר, והוא ריק ממים.

נוסף על הפעולה להעברת מים – דלי X– שפוך מים ל(דלי Y), התווספה גם פעולה לריקון דלי: דליX – רוקן מים.

**כתבו תוכנית** שבסופה יהיו בדלי השני יהיו 2 ליטר מים בדיוק. לרשותכם הפעולות הבאות:

**תזכורת:** בפעולה שבה שופכים מים מדלי לדלי לא ניתן להגדיר כמה מים להעביר – הפעולה מעבירה מים ככל שניתן עד שהדלי השני מתמלא או שהדלי הראשון מתרוקן.

כלומר המצב ההתחלתי יהיה: והמצב הסופי יהיה: 

**פתרון:**

אחד הכלים שבהם נוכל להשתמש כדי לתכנן פתרון הוא טבלה שתתאר את המצב של מערכת הדליים בכל שלב.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| כמות המים בדליים לאחר ביצוע הפעולה | | |  |  |  |
| בדלי 1 (מתוך 10) | בדלי 2 (מתוך 4) | בדלי 3 (מתוך 5) | | | הפעולה לביצוע |
| 10 | 0 | 0 | | | דלי 1 מלא במים |
| 5 | 0 | 5 | | | מעבירים מים מדלי 1 לדלי 3 |
| 1 | 4 | 5 | | | מעבירים מים מדלי 1 לדלי 2 |
| 1 | 0 | 5 | | | מרוקנים את דלי 2 |
| 1 | 4 | 1 | | | מעבירים מים מדלי 3 לדלי 2 |
| 1 | 0 | 1 | | | מרוקנים את דלי 2 |
| 1 | 1 | 0 | | | מעבירים מים מדלי 3 לדלי 2 |
| 0 | 2 | 0 | | | מעבירים מים מדלי 1 לדלי 2 |
|  |  |  | | | הגענו למצב הרצוי |

אם אתם מעדיפים להשתמש בשפה חופשית ולהימנע משימוש בטבלה, לרשותכם פתרון אפשרי בשפה מקוצרת:

דלי1 – שפוך מים ל(דלי3) // כעת נשארו 5 ליטר מים בדלי1 ומצב הדליים הוא: 5, 0, 5.

דלי1 – שפוך מים ל(דלי2) // כעת נשארו 1 ליטר מים בדלי1 ומצב הדליים הוא: 1, 4, 5.

דלי2 – רוקן מים // לא נשארו מים בדלי2 ומצב הדליים הוא: 1, 0, 5.

דלי3 – שפוך מים ל(דלי2) // כעת נשארו 1 ליטר מים בדלי3 ומצב הדליים הוא: 1, 4, 1.

דלי2 – רוקן מים // לא נשארו מים בדלי2 ומצב הדליים הוא: 1, 0, 1.

דלי3 – שפוך מים ל(דלי2) // כעת נשארו 1 ליטר מים בדלי3 ומצב הדליים הוא: 1, 1, 0.

דלי1 – שפוך מים ל(דלי2) // כעת נשארו 0 ליטר מים בדלי1 ומצב הדליים הוא: 0, 2, 0.

#### חלק ב

ברשותכם שני דליים מלאים ב-10 ליטר מים כל אחד, וזו גם הקיבולת המקסימלית של כל אחד מהם. **הציעו** אלגוריתם שבסופו יהיו 2 ליטר מים בדלי שקיבולתו 4 ליטר ושני ליטר מים בכלי שקיבולתו 5 ליטר. יש להקפיד לא לרוקן את הדליים כלל, כלומר שבסוף התהליך לא תאבד אף טיפת מים.

##### פעולות מתמטיות מתקדמות בעזרת המחלקה Math

בפרק זה ראינו כי אופרטור החיבור, אופרטור החיסור ואופרטור הכפל פועלים על ביטויים חשבוניים באופן המוכר לנו מלימודי המתמטיקה. עם זאת, ראינו גם שאופרטורים יכולים לפעול בצורה שונה בהתאם לטיפוסי הנתונים שעליהם הם פועלים, כמו בדוגמת חלוקת השלמים ושרשור המחרוזות.

ישנן פעולות מתמטיות נפוצות רבות, וכדי לאפשר למתכנתים להשתמש בהן בקלות הן רוכזו במחלקה מיוחדת שנקראת **Math.** מחלקה זו אינה משמשת לבניית עצמים אלא מרכזת פעולות שימושיות. כדי להפעיל פעולה מהמחלקה יש לקרוא לפעולה בצורה הבאה:

Math.\_\_\_\_\_\_\_\_\_\_\_\_

במקום הקו יופיע שם הפעולה. שימו לב שלאחר שם המחלקה יש נקודה, ורק אחריה נרשמת הפעולה הרצויה.

למשל, להדפסת ערך המקסימום מבין שני מספרים ניתן לרשום את ההוראה הבאה:

System.out.println(Math.max(12, 45));

הפעולה הרצויה תמיד תסתיים בסוגריים (). לרוב בתוך הסוגריים יירשמו ערכים שעליהם הפעולה תתבצע והם נקראים **ערכים פרמטריים**.

נציג כאן כמה פעולות לדוגמה של המחלקה Math. לא נציג את כל הפעולות, שכן אנו לא זקוקים כעת לרובן. בהמשך תוכלו לקרוא את התיעוד של המחלקה או לחפש ברשת האינטרנט פרטים לגבי פעולות רצויות.



### משימה 15: מציאת ערך המקסימום בין שני ערכי משתנים

התבוננו בשתי התוכניות הבאות **וענו** על השאלות:

1. מהו ההבדל בין שתי התוכניות?
2. העריכו מה יהיה ההבדל בפלט התוכנית?  
   **בדקו** את הערכתכם בתוכנית מחשב.

תוכנית א:

public static void main(String[] args) {

int number2 = 100;

int number1 = 93;

System.out.println(Math.max(number1, number2));

}

C#

static void Main(string[] args)

int number2 = 100;

int number1 = 93;

Console.WriteLineln(Math.Max(number1, number2));

}

תוכנית ב:

public static void main(String[] args) {

int number2 = 100;

double number1 = 93.8;

System.out.println(Math.max(number1, number2));

}

C#

static void Main(string[] args) {

int number2 = 100;

double number1 = 93.8;

Console.WriteLineln(Math.Max(number1, number2));

}



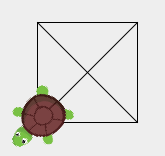
##### משימה 15: מציאת מספר מקסימלי ומינימלי (המשך)

1. **פתחו** אלגוריתם אשר יקלוט מהמשתמש שני מספרים וידפיס את המספר **המקסימלי** ביניהם.   
   **שימו לב!** הקפידו לפנות אל המשתמש בבקשה להכניס את המספרים, ולאחר הקליטה והחישוב להדפיס את התוצאה בליווי הודעה מתאימה.  
   **ממשו** את האלגוריתם באמצעות תוכנית מחשב.
2. **פתחו** אלגוריתם אשר יקלוט מהמשתמש שני מספרים וידפיס את המספר ה**מינימלי** ביניהם.   
   גם כאן יש להקפיד לפנות אל המשתמש בבקשה להכניס את המספרים, ולאחר הקליטה והחישוב להדפיס את התוצאה בליווי הודעה מתאימה.  
   **ממשו** את האלגוריתם באמצעות תוכנית מחשב.

**הערה:** את הפעולה המתאימה במחלקה Math לחישוב מינימום עליכם למצוא לבד. תוכלו לחפש אותה ברשת האינטרנט או לנסות ניחוש מושכל לאור היכרותכם עם הפעולה Math.max. בהמשך נכיר אפשרויות נוספות למציאת פעולות רצויות במחלקה Math.



##### משימה 16: שרטוט אלכסון לריבוע



**פתחו** את האלגוריתם הבא:

**קלט**: אורך צלע רצוי.

**פלט**: ציור של ריבוע בעל אורך צלע מתאים (שהוזן בקלט) ושני האלכסונים.  
 והדפסה ב-console של אורך האלכסונים.

הערה: תוכלו לבחור את כיוון הציור כרצונכם

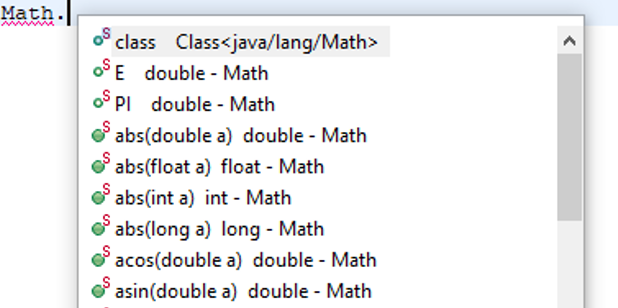
**פתרון**

כהרגלנו, תחילה נתכנן את הפתרון. זוהי בעיה גיאומטרית – האלכסון הוא היתר במשולש ישר-זווית שווה-שוקיים שאורך כל ניצב בו שווה ל-side. נוכל לחשב את אורכו בעזרת משפט פיתגורס: אורך היתר הוא שורש הסכום של ריבועי הניצבים.

אפשר לחפש את הפעולה הרצויה בממשק המחלקה. קל למצוא אותה באינטרנט אם תכתבו את מילות החיפוש הבאות: java math api. המילה api היא ראשי התיבות של Application Programming Interface, ומעלה את פירוט הפעולות של מחלקה מסוימת.

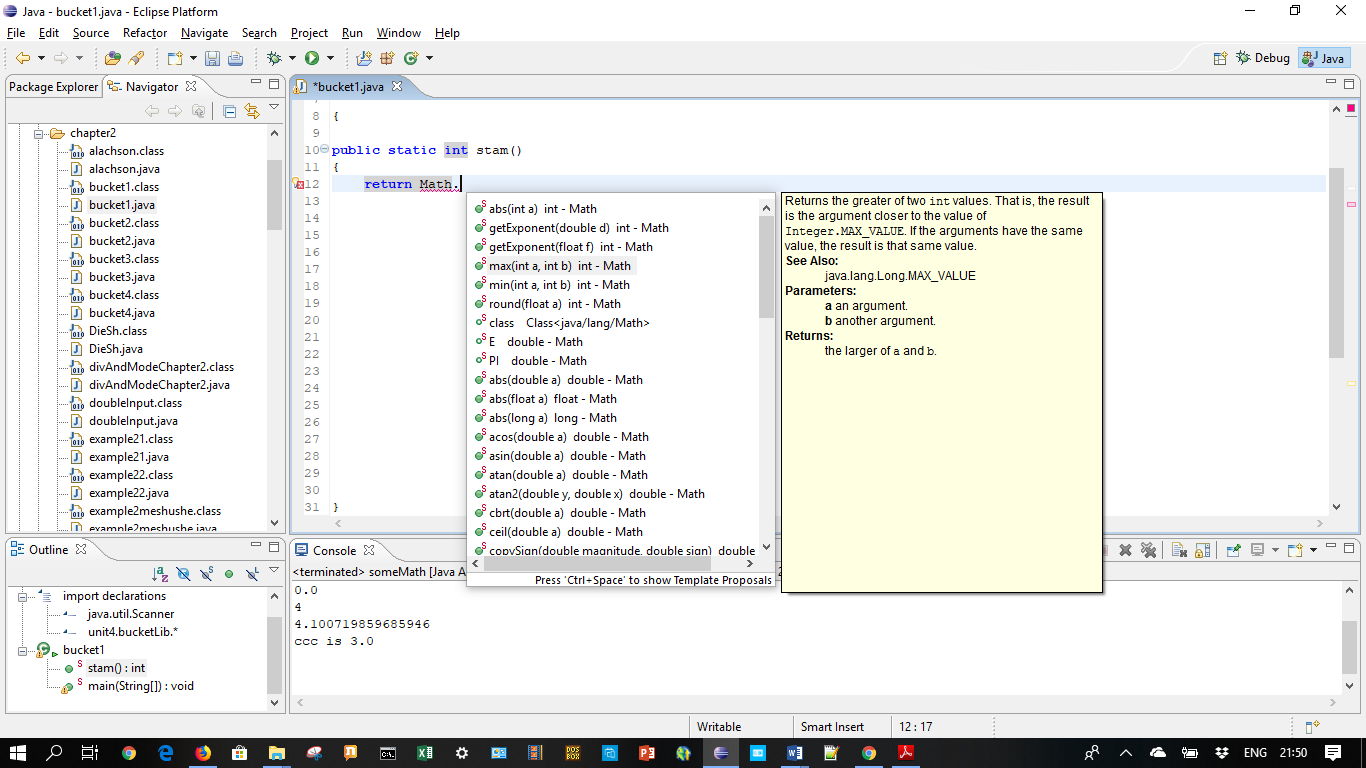
אפשר גם ללחוץ [כאן](https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html).

(הקישור הוא <https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html> )

אפשרות נוספת היא לחפש את הפעולה שורש ישירות בסביבת העבודה: כתבו את המילה Math בסביבת העבודה ואחריה נקודה. ייפרש לפניכם התפריט הבא:

בתפריט מוצגות כל הפעולות שאפשר לבצע בעזרת המחלקה. תוכלו לגלול מטה עד שתמצאו את הפעולה המתאימה.

אם תציבו את סמן העכבר על פעולה מסוימת, תראו פירוט שלה מימין.





##### משימה 17: קו בין שתי נקודות

**פתחו** אלגוריתם אשר יקלוט מהמשתמש ארבעה נתונים: שני הנתונים הראשונים מיצגים ערכי X ו-Y של נקודה במישור, ושני הנתונים הבאים מייצגים ערכי X ו-Y של נקודה אחרת במישור.

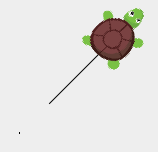
המיקום ההתחלתי של הצב הוא ראשית הצירים (0,0). הצב יצייר את הקו בין הנקודות שאת ערכיהן הכניס המשתמש.

למשל, אם הוכנסו הערכים 30,30,100,100 – יצויר הקו שבכותרת המשימה:

(**רמז:** הקושי בתוכנית יהיה מציאת הזווית המתאימה לפנייה במעלות ואורך הקו)

**ממשו** את האלגוריתם כתוכנית מחשב.

הנקודה מייצגת את ראשית הצירים



### המרה בין טיפוסים

בפעולות מתמטיות שונות הערך המוחזר מהן הוא מטיפוס ממשי, ואנחנו מעוניינים בתוצאה מטיפוס שלם.

למשל, הפעולה המתמטית floor מחזירה את הערך הממשי הגדול ביותר שקטן או שווה לערך שנשלח כפרמטר ושווה בערכו למספר שלם.

ההוראה הבאה:

System.out.println(Math.floor(4.8));

Console.WriteLineln(Math.Floor(4.8));

תדפיס את הערך 4.0, שכן זהו המספר השווה למספר שלם הקטן (או שווה) מהערך ששלחנו. הפעולה בהגדרתה מחזירה מספר ממשי.

רגע חושבים!

מה תדפיס ההוראה הבאה?

System.out.println(Math.floor(4));

Console.WriteLineln(Math.Floor(4));

במקרה של הדפסה נוכל אולי להתפשר על הדפסה עם נקודה עשרונית, אבל לעיתים נרצה לשמור את הערך המוחזר במשתנה מטיפוס אחר.

לדוגמה אם נרצה לחשב ריבוע של מספר שלם, נוכל להיעזר בפעולה המתמטית חזקה:

System.out.println(Math.pow(4, 2));

Console.WriteLineln(Math.Pow(4, 2));

תוצאת החישוב של 4 בחזקת 2 תהיה 16.0, כלומר מספר ממשי. הצבה של מספר זה במשתנה מטיפוס שלם תעלה שגיאה.

אם אין באפשרותנו להשתמש במשתנה מטיפוס ממשי, נבצע המרה של הערך המוחזר לטיפוס שלם.

כדי להמיר ערך מטיפוס לטיפוס נשתמש בפעולת המרה (casting). הפעולה מתבצעת באמצעות הוספת הטיפוס המומר בתוך סוגריים לפני ההשמה למשתנה. למשל:

int money = (int) Math.pow(4, 2);

פעולה זו אפשרית, והערך 16 יושם לתוך המשתנה money.

הכיוון ההפוך אפשרי גם הוא:

double average = (double) 4 / 5;

שימו לב שההמרה היא בעלת סדר עדיפות גבוה יותר מהחילוק: המספר השלם 4 מומר למספר הממשי 4.0, ורק אז מתבצעת החלוקה. ללא ההמרה הייתה מתבצעת חלוקת שלמים והתוצאה הייתה 0. המשתנה average הוא אומנם ממשי, אבל כזכור החלוקה מתבצעת לפני ההשמה, ולכן הערך המושם היה משתנה (במקרה זה בהמרה אוטומטית) ל-0.0. אך כיוון שההמרה מתבצעת לפני החילוק, 4 הופך ל-4.0, ואז כבר לא מתבצעת חלוקת שלמים והתוצאה המתקבלת היא 0.8.

כאמור, לסוגריים יש עדיפות גבוהה יותר מפעולת ההמרה. שערו ובדקו מה יושם בכל מקרה:

double average = ((double) 4) / 5;

double average = (double)(4 / 5);



##### משימה 18: ממוצע

במסגרת הדיון האם כדאי להפיק עונה חדשה לסדרה "גאליז", הוחלט לחשב את אחוזי הצפייה הממוצעים בעונה האחרונה של התוכנית. בעונה זו שודרו שישה פרקים, ואחוזי הצפייה בכל פרק ידועים למפיקים.

התבקשתם לעזור למפיקי התוכנית לחשב את אחוזי הצפייה הממוצעים בעונה.

**פתחו** את האלגוריתם הבא:

**קלט**: אחוזי הצפייה בכל אחד מששת הפרקים בעונה האחרונה.

**פלט**: הדפסה (ב-console) של אחוזי הצפייה הממוצעים לעונה האחרונה.

**ממשו** את האלגוריתם כתוכנית מחשב.

### מספר אקראי

במשחקים רבים, כמו במשחק הזיכרון שתכננתם בפרק הקודם, יש צורך בבחירות אקראיות. לדוגמה, יש להחליט כיצד לערבב את הקלפים.

במחלקה Math יש פעולה שמגרילה מספרים אקראיים. היא מחזירה מספר ממשי גדול או שווה ל-0.0 וקטן ממשי מ-1.0. כלומר אף פעם לא נעלה מעל מספר 1!

למשל אם תריצו תוכנית הכוללת את השורות הבאות:

double rndNumber=Math.random();

System.out.println(rndNumber);

תקבלו הדפסות מעניינות כמו:

0.24455210422745033

או:

0.41246522310552103

בכל פעם יוגרל מספר אחר.

אבל המטרה שלכם בדרך כלל תהיה להגריל מספרים שלמים בתחום נתון. למשל להגריל תוצאה של הטלת קובייה, כלומר מספר בין 1 ל-6.



##### משימה 19: הטלת קובייה

**כתבו** תוכנית המגרילה מספר בעזרת הפעולה Math.random() ומדפיסה מספר שלם בין 1 ל-6 (כולל).

המטרה: המרת מספר ממשי גדול מ-0 וקטן מ-1 למספר שלם בין 1 ל-6 (כולל).

נחלק את המשימה שלנו לשני חלקים: הראשון יהיה להרחיב את גבולות המספרים, כלומר להמיר את המספר הממשי בין 0 ל-1 (לא כולל) למספר ממשי מספרים בין 0 ל-6 (לא כולל). התוצאה תהיה 6 מספרים שלמים שונים (0,1,2,3,4,5).

הפתרון שנבחר בו יהיה הכפלה של המספר המוגרל במספר גדול מ-1 שמתאים לנו. אם נכפיל ב-6 נקבל בהכרח מספר ממשי גדול או שווה ל-0 וקטן מ-6 – למשל, הכפלה ב-6 של המספר בדוגמה הראשונה ייתן תוצאה של 1.46731262536470. נוכל להחליט שמספר בין 1 ל-2 ייצג את הערך 1 או את הערך 2. בהתאם לכך נמשיך בתכנון. בדוגמה זו נחליט שהערך שהתקבל משמעו שהמספר 1 התקבל בזריקת הקובייה. בהמשך יובהר מדוע בחרנו באפשרות זו.

הכפלה ב-6 של המספר בדוגמה השנייה ייתן את התוצאה 2.47479133863312618, ומכאן נבין שהמספר 2 התקבל בזריקת הקובייה.

נותר רק להמיר את המספר שמתקבל למספר שלם.

ההוראה הבאה תשים במשתנה dice (שמשמעו צורת היחיד של קוביות) מספר ממשי הגדול או שווה ל-0 אך בהכרח קטן מ-6.

double dice = Math.random() \* 6;

ההמרה לשלם תיראה כך:

int intDice = (int) dice;

שים במשתנה intDice מספר שלם בין 0 ל-5.

בפעולה זו ביצענו הרחבה של חלון המספרים שהוגרלו ל"רוחב" הנכון, אבל אנחנו לא מתחילים מהמספר הנכון. מה שחסרה לנו זו הזזה.

נזיז את התוצאה ב-1 ונקבל את הגבולות הנכונים:

int intDice = (int)(Math.random() \* 6) + 1;

נכליל את התוצאה למספרים בין 1 למספר כלשהו:

כדי להגריל מספר שלם בתחום בין 1 למספר כלשהו max. ניתן לבצע את ההוראה הבאה:

int intDice = (int)(Math.random() \* max) + 1;

חשוב להדגיש שההמרה למספר שלם מתבצעת רק לאחר ההכפלה, אחרת המספר שהמחלקה Math תגריל יומר למספר השלם 0!

נכליל את התוצאה לתחום מספרים כלשהו:

כדי להגריל מספר שלם בתחום בין מספר כלשהו min למספר כלשהו max. יש לבצע את ההוראה הבאה:

int intNumber = (int)(Math.random() \* ) max - min) + min;

ההכפלה מרחיבה את ההגרלה המקורית ל"רוחב" הרצוי וההוספה מזיזה את התוצאות לתחילת התחום.



##### משימה 20: הגרלת מספר זוכה

גלריית CS לאומנות מודרנית מכרה 500 כרטיסים לאירוע הפתיחה. הוחלט להגריל חופשה בין כל כרטיסי הכניסה שנרכשו. הכרטיסים ממוספרים מהמספר 1001 ועד המספר 1500. **כתבו** תוכנית המגרילה מספר בעזרת הפעולה Math.random() ומדפיסה מספר שלם בין 1001 ל-1500 כולל.

##### סיכום פרק מושגי יסוד

* **משתנה** או **תכונה** הוא מקום בזיכרון בעל שם ו**טיפוס** מוגדרים. המשתנה מאפשר לנו לשמור (לרשום) בו ערך לשימוש עתידי.
* **טיפוס נתונים** הוא מאפיין של קבוצת הערכים שניתן להשים במשתנה מסוים.
* **שמות משמעותיים** למשתנים הם הכרחיים לקריאות של התוכנית ולהבנת ייעוד המשתנים.
* **הוראת ההשמה** מבוצעת על ידי אופרטור ההשמה ומאפשרת לנו לשמור נתונים לשימוש בהמשך התוכנית.
* **אתחול** המשתנים הוא מתן ערך התחלתי למשתנה. זוהי פעולה הכרחית למקרה שנרצה להשתמש בהם בהוראות שונות ובפרט ב**ביטוי חשבוני**.
* **ביטוי חשבוני** מורכב ממספרים או משתנים חשבוניים ומאופרטורים חשבוניים. תוצאת הביטוי החשבוני היא ערך חשבוני מטיפוס מסוים הניתן להשמה במשתנים מתאימים.
* **אופרטור** הוא פעולה שפועלת על איברים בקבוצה מסוימת ותוצאתה היא איבר ששייך גם הוא לקבוצה.
* **האופרטור %** הוא אופרטור שתוצאתו היא שארית **חלוקת השלמים** של שני מספרים שלמים. הוא מסומן בג'ווה וב-C# בסימן **%**.
* **הוראות קלט** הן הוראות המאפשרות לנו לקלוט נתונים מהמשתמש והן משתנות בהתאם לטיפוס נתון הערך הנקלט.
* **הוראות פלט** מדפיסות הודעות המאפשרות תקשורת עם המשתמש.
* **מחרוזת** היא עצם שאפשר לכתוב באופן פשוט ומקוצר. הכרנו את פעולות המחלקה מחרוזת, אבל ראינו שכשאופרטור החיבור פועל על מחרוזות הוא מבצע שרשור.
* המחלקה **Math** מכילה פעולות מתמטיות נפוצות וקבועים מתמטיים נפוצים.
* סיכום כל הפעולות והקבועים במחלקה תמצאו בקישור הבא:
* <https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>

לפניכם טבלה המסכמת פעולות נפוצות עם המחלקה Math.

המחלקה המתמטית Math מכילה עוד פעולות מתמטיות רבות אשר נתונות לשימושנו, וכמוה יש מחלקות נוספות שנוכל להשתמש בהן לפעולות עזר מסוגים שונים.

###### פעולות שימושיות מהמחלקה המתמטית **Math**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **הפעולה** | **תיאור הפעולה** | **טיפוסי פרמטרים** | **טיפוס ערך מוחזר** | **דוגמה** | |
| **הפעולה** | **הערך המוחזר** |
| abs (num) | ערך מוחלט | שלם ממשי | שלם ממשי | Math.abs(63)  Math.abs(-12.7) | 63  12.7 |
| sqrt (num) | שורש ריבועי | ממשי | ממשי | Math.sqrt(6.25) | 2.5 |
| pow (num1, num2) | חזקה num1num2 | ממשי, ממשי | ממשי | Math.pow(3,2) | 9.0 |
| min (num1, num2) | הקטן מבין השניים | שלם, שלם ממשי, ממשי | שלם ממשי | Math.min(3,8)  Math.min(8.0,8.8) | 3  8.0 |
| max (num1, num2) | הגדול מבין השניים | שלם, שלם ממשי, ממשי | שלם ממשי | Math.max(3,8)  Math.max(8.0,8.8) | 8  8.8 |
| round (num) | עיגול מספר ממשי | ממשי | שלם | Math.round(7.9) | 8 |

בנוסף לפעולות המתוארות בטבלה זו שייכות למחלקה Math פעולת רבות נוספות. אתם מוזמנים לחפש ברשת ולעיין בממשק של המחלקה המתמטית בשפת **Java** (java.lang.Math) ולמצוא פעולות נוספות השייכות למחלקה זו.

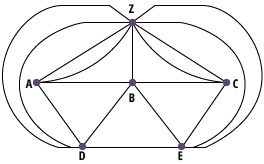
### חמישה בחמש

לפניכם רשימת נושאים לעבודת חקר קצרה.  
**בחרו** את אחד מהנושאים ברשימה, ו**הכינו** מצגת בת חמישה שקפים.  
לרשותכם חמש דקות להציג בפני הכיתה את הנושא הנבחר.

1. השימוש בצב להדגמות קיים בשפות נוספות וגם בשיעורי למידה עצמית ברשת. למשל באתר [אקדמיית הצב](https://turtleacademy.com/index/he) בכתובת: <https://turtleacademy.com/index/he>

מהיכן הגיע הצב המקורי...?

1. בפרק נתקלתם בציור של משולש ישר-זווית. הסבירו מהו משפט פיתגורס, מהי שלשה פיתגוראית וכיצד הן רלוונטיות למדעי המחשב.
2. בפרק הוזכרו המושגים קלט ופלט, ובשני המקרים המקור והיעד היו ה-console. האם תוכלו לחשוב על אמצעי קלט נוספים? אמצעי הפלט שלנו הוא המסך, חפשו סוגי קלט מעניינים נוספים. הרחיבו על מושגים אלו.
3. במשולש שווה-צלעות כל זווית היא בת 60 מעלות. ציירנו משושה משוכלל שבו כל זווית היא בת 120מעלות. מה הקשר בין מספר הצלעות לערך הזווית? האם יש קשרים דומים במצולעים אחרים?
4. האם טיפוס נתונים מסוג מחרוזת יכול להיות אינסופי בגודלו? האם ישנם טיפוסי נתונים אחרים שיכולים להיות אינסופיים באורכם? הסבירו מדוע.
5. ציור במשיכת קולמוס אחת משמעו ציור מבלי להרים את העט מהנייר ומבלי לחזור על קו יותר מפעם אחת. ישנם שני כללים שמגדירים האם צורה ניתנת לציור במשיכת קולמוס אחת. בררו מהי בעיית הגשרים של קניגסברג, מי פתר אותה, ואילו חוקים הוא קבע כדי לקבוע האם ניתן לצייר ציור במשיכת קולמוס אחת, ובפרט האם ניתן לצייר את הצורה הבאה:



1. בפרק הוזכרו מספרים אקראיים. ניתן לרכוש באמזון ספר שנקרא מיליון מספרים אקראיים. חקרו קצת והבינו מדוע נכתב ספר זה?  
   <https://www.amazon.com/Million-Random-Digits-Normal-Deviates/dp/0833030477>
2. בבעיית הציור במשיכת קולמוס הוזכר מתמטיקאי. ישנו היום פרויקט לחובבי תכנות שנקרא על שמו, פרויקט \_\_\_\_\_\_. האם תוכלו לפתור את השאלה הראשונה בפרויקט?  
   הציגו פרויקט זה בכיתה.
3. חלוקה באפס מוגדרת על ידי תלמידים רבים כבח"ם – ביטוי חסר משמעות. לדעתנו עדיף להגדיר אותה כבל"מ – ביטוי לא מוגדר. בכל מקרה חלוקה באפס בתוכנית המחשב שלכם תגרור שגיאה. חקרו קצת נושא זה, ובררו מה הקשר בין נקודה סינגולרית במתמטיקה לבין נקודת הזמן של הסינגולריות הטכנולוגית.
4. מה הקשר בין שורש של מינוס 1 ( ) לבין משולש ישר-זווית?

#### פרק 3: ביצוע מותנה

##### הקדמה

בפרק הקודם הכרנו מספר מושגים בסיסיים הנדרשים לביצוע מגוון משימות באמצעות המחשב, למשל ציור צורות גיאומטריות או פתרון חידות. בתוכניות שהוצגו לפנינו, התוכנית ביצעה פעולות בזו אחר זו לפי הסדר שבו נכתבו מבלי אפשרות לדלג עליהן. בפרק זה נכיר מהו משפט תנאי או פקודת תנאי, שבאמצעותה ניתן להורות למחשב "לבחור" כיצד לפעול במצבים שונים.

חשוב לציין שהתנאי הוא כלי אינטואיטיבי בקבלת החלטות בחיי היום יום וכולנו מנוסים בו. עליכם רק להתאמן בתרגום צורת החשיבה הזו לרישום אלגוריתמי ולכתיבה נכונה בשפת מחשב.



משימה 1: הבית החכם

כיום אפשר לשלוט על דברים רבים בבית באמצעות אפליקציות: הדלקת מזגן, כיבוי אורות, הדלקת דוד ועוד. במשימה הקרובה תפתחו אלגוריתם למתג תאורה בבית חכם. נסמן מצב שבו האור דולק בבית במספר 1, ומצב שבו האור כבוי במספר 0.

המתג החכם מקבל מהמשתמש (או מחיישנים) מספר 0 או 1, והופך אותו. כלומר:

קלט: 0 או 1 (לא יהיה קלט אחר, אין צורך לבדוק זאת)

פלט: אם הקלט הוא 1, יודפס 0. אם הקלט הוא 0, יודפס 1.

נפתח את האלגוריתם בשפה חופשית:

קלוט מספר

אם המספר שנקלט הוא 1, הדפס 0

אחרת הדפס 1 (ידוע שיש רק שתי אפשרויות בשאלה זו)

איך ניגשים לתרגם זאת לשפת המחשב?

### ביצועמותנה

### זוהי התבנית הבסיסית של ביצוע מותנה:

אם מתקיים **תנאי מסוים**, נבצע **סדרת פעולות**.

אחרת נבצע **סדרת פעולות (כנראה אחרת)**.

נניח שקלטתם את המספר למשתנה light. **השלימו** את ההוראות החסרות:

if (light == 1)

{

סדרת פעולות (יכולה להיות גם פעולה בודדת) \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

}

else

{

סדרת פעולות (כנראה אחרת) \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

}

**פתרון:**

בהתאם לאלגוריתם, התשובה צריכה להיות

|  |
| --- |
| System.out.println(0); |

**C#**

|  |
| --- |
| Console.WriteLine(0); |

ו-

|  |
| --- |
| System.out.println(1); |

**C#**

|  |
| --- |
| Console.WriteLine(1); |

בהתאמה.

התבוננו בחלקי משפט התנאי:

* התנאי תחום בסוגריים עגולים.
* סדרת הפעולות התחומה בסוגריים מסולסלים נקראת גם **בלוק הפקודות** לביצוע.
* ההשוואה בין המספר שנקלט למספר 1 מסומנת באופרטור השוואה, הסימן == .

**שימו לב:** אופרטור ההשוואה בודק האם הערכים שמשני צדדיו שווים. זאת בשונה מאופרטור ההשמה = שמשים ערך בתוך משתנה, כפי שראיתם לדוגמה בהוראה light = 1.

הביטוי light == 1 מכונה **ביטוי בוליאני**. ביטוי זה יכול לקבל אחד מבין שני ערכים בוליאניים אפשריים – **אמת** או **שקר**, ובסביבת העבודה **true** או **false**. ביטוי בוליאני יכול להשוות בין ערכים, משתנים וביטויים חשבוניים.

לפניכם טבלה עם אופרטורים של השוואה שבעזרתם ניתן לכתוב ביטויים בוליאניים, עם דוגמאות נלוות:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **סימן ההשוואה ב-C# או בג'אווה** | **סימן ההשוואה המקובל במתמטיקה** | **משמעות סימן ההשוואה** | **דוגמה ב-C# או בג'ווה** | **דוגמה במתמטיקה** |
| == | = | שווה | X == 5 | X = 5 |
| != | ≠ | שונה | X != y | x ≠ y |
| < | < | קטן | X < 2 | X < 2 |
| <= | ≥ | קטן או שווה | X <= 1 | X ≤ 2 |
| > | < | גדול | Y > 0 | Y > 0 |
| >= | ≤ | גדול או שווה | Y >= 8 | Y ≥ 8 |

ניתן לרשום את מבנה התנאי שהוצג גם בצורה הבאה:

אם ערכו של **ביטוי בוליאני** הוא אמת, נבצע **בלוק פקודות**.

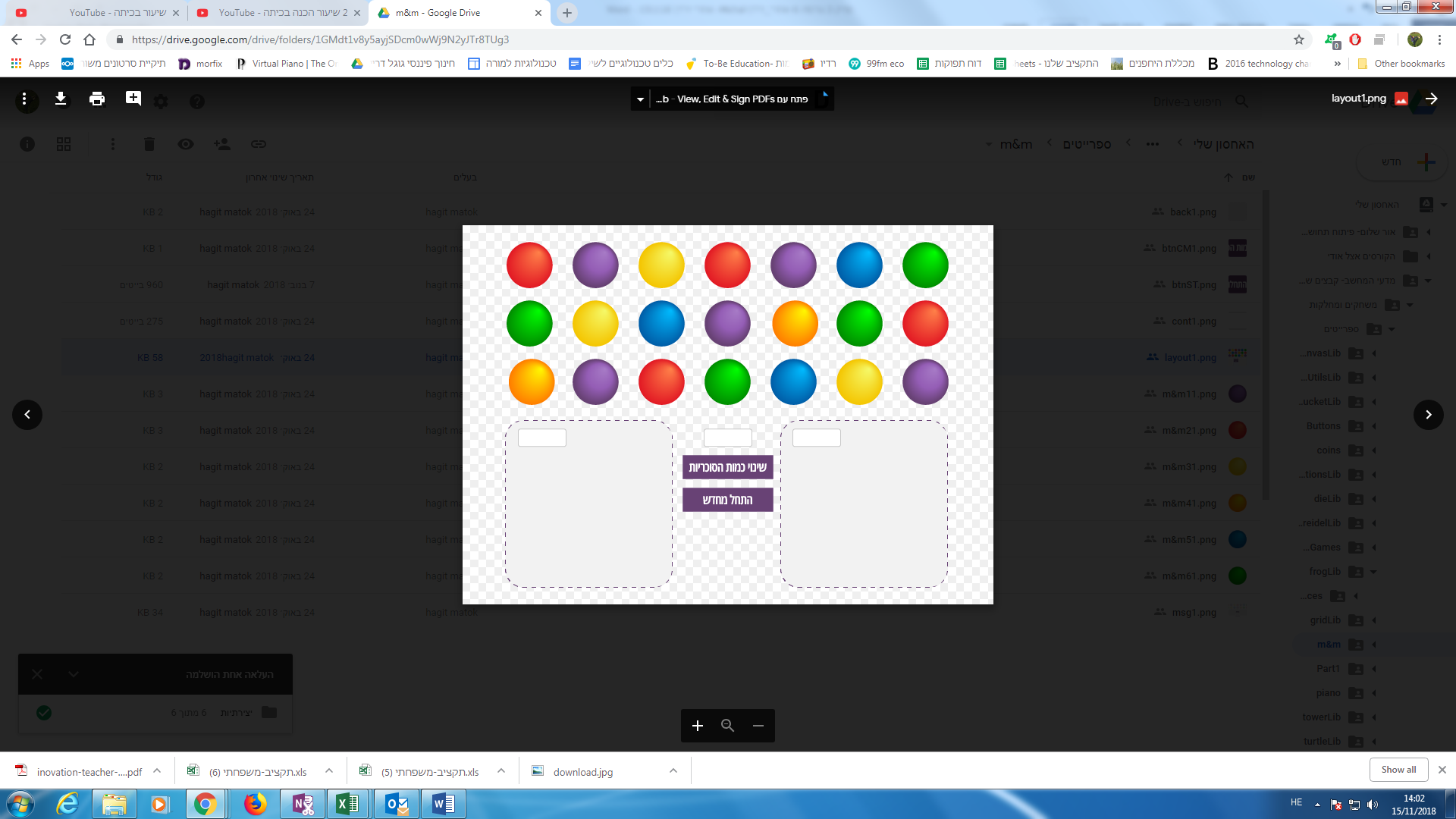
אחרת נבצע **בלוק פקודות (כנראה אחרות)**.

### שימוש בתנאים בתוכנית מחשב

תנאים מאפשרים לנתב את זרימת התוכנית שכתבנו. במקום לבצע רצף הוראות עוקבות, ניתן להורות אלו הוראות לבצע ואלו לא לבצע בהתאם לקיומו או אי-קיומו של תנאי.

להעמקת ההבנה בשימוש בתנאים, נכיר משחק חדש ששייך למשפחת משחקי נים.

אלו חוקי המשחק:



1. בערמה יש 21 סוכריות.
2. במשחק משחקים שני שחקנים.
3. המשחק מתנהל בתורות ולא ניתן לוותר על תור.
4. במשחקון השני והלאה מתחיל השחקן שניצח במשחקון הקודם. במשחקון הראשון מתחיל השחקן המבוגר מבין השניים.
5. בכל תור מותר לשחקן לקחת סוכריה אחת, שתי סוכריות או שלוש סוכריות בלבד.
6. השחקן שלוקח את הסוכרייה האחרונה מפסיד.

תוכלו להתנסות במשחק בקישור הבא

**רגע חושבים!**

יש לשחק במשחק, להתנסות בו ולנתח את המצבים השונים לפני כתיבת התוכנה.



משימה 3.3: שאלה הדורשת ניתוח – כתיבת תוכנת עזר

**כתבו** תוכנה אשר מייעצת לשחקן כמה סוכריות כדאי לו לקחת בכל מצב נתון. לפני שמתחילים לחשוב על האלגוריתם, מומלץ לארגן ערמה של סוכריות או פיסות נייר ולשחק במשחק כמה פעמים.

לאחר מכן **ארגנו** את התובנות שלכם בטבלה הבאה:

|  |  |  |
| --- | --- | --- |
| **מספר הסוכריות בערמה** | **כמה כדאי לי לקחת** | **הסבר קצר** |
|  | חייב לקחת 1 | הפסדתי |
|  | 1 | היריב יישאר עם הסוכרייה האחרונה ויהיה חייב לקחת אותה |
|  | 2 | היריב יישאר עם הסוכרייה האחרונה ויהיה חייב לקחת אותה |
|  | 3 | היריב יישאר עם הסוכרייה האחרונה ויהיה חייב לקחת אותה |
|  | לא ניתן לוודא ניצחון | אם היריב יודע לשחק הוא ינצח |
|  | 1 | היריב יישאר עם 5 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| . |  |  |
| . |  |  |
|  |  |  |

**רגע חושבים!**

אם שני השחקנים יודעים את חוקיות המשחק, מי ינצח? זה שמתחיל או זה שמסיים?

לאחר שהתנסיתם במשחק מספר פעמים, בוודאי גיליתם שהשחקן המתחיל הוא זה שיפסיד. האם זהות המנצח תשתנה אם מספר הסוכריות על השולחן יהיה שונה?

התנסו במשחק ובדקו כיצד הכלים של מדעי המחשב מסייעים במענה על שאלה זו. באמצעות הניתוח שערכתם, עליכם לגשת לכתיבת האלגוריתם לתוכנית המייעצת. איך עושים זאת?

לפניכם שלושה פתרונות אפשריים:



משימה 4: אפשרות ראשונה

**פתחו** אלגוריתם שמשתמש באופן ישיר במבנה הבקרה של תנאי כפי שלמדתם עד כה.

**שימו לב** כי האלגוריתם צריך לייעץ לשחקנים רק אם יש 21 סוכריות או פחות. לכן, עליכם לכתוב תנאי בתוכנה שמבקש מהשחקנים להזין מספר חדש אם כתבו מספר גדול מ-21.

**פתרון:**

מצב התחלתי: המצב ההתחלתי נקבע על פי מספר הסוכריות בערמה. נשים לב כי ישנם מספרים "בעייתיים" שעבורם כל בחירה של השחקן תביא להפסד, ואחרים יאפשרו ניצחון. לכאורה הפתרון ברור: לכל מצב (מספר סוכריות בערמה) יש לכתוב מהי הבחירה אשר תוביל לניצחון.

תכנון רצף ההוראות בשפה חופשית:

אם בערמה סוכריה אחת, קח אחת והפסדת.

אם בערמה 2 סוכריות, קח אחת.

אם בערמה 3 סוכריות, קח 2.

אם בערמה 4 סוכריות, קח 3.

אם בערמה 5 סוכריות, קח אחת ועשה פרצוף מתעמק, מתוך תקווה שהיריב לא ישים לב שהוא יכול לנצח...

אם בערמה 6 סוכריות, קח אחת.

וכך הלאה עד ל-21.

**רגע חושבים!**

האם יש טעות באלגוריתם? האם חסר בו משהו?

שימו לב כי בתכנון בשפה חופשית של האלגוריתם, אין צורך בחלק השני של הוראת התנאי – אחרת.

בשפה חופשית, גם בסביבת העבודה אפשר לוותר על החלק השני.

במקרה זה מספר הסוכריות נקלט למשתנה pile, ולכן עליכם לרשום:

if (pile == 1) {

System.out.println(“Take the last candy – but you have lost…”);

}

if (pile == 2) {

System.out.println(“Take 1 candy“);

}

if (pile == 3) {

System.out.println(“Take 2 candies”);

}

if (pile == 4) {

System.out.println(“Take 3 candies”);

}

if (pile == 5) {

System.out.println(“Take 1 candy and hope for the best…”);

}

C#

if (pile == 1) {

Console.WriteLine(“Take the last candy – but you have lost…”);

}

if (pile == 2) {

Console.WriteLine(“Take 1 candy“);

}

if (pile == 3) {

Console.WriteLine(“Take 2 candies”);

}

if (pile == 4) {

Console.WriteLine(“Take 3 candies”);

}

if (pile == 5) {

Console.WriteLine(“Take 1 candy and hope for the best…”);

**השלימו** את עשרת המספרים הבאים בעצמכם.

כזכור, בכל במצב שבו יש בערמה יותר מ-21 סוכריות, התוכנית שכתבתם לא תספק פתרון. לכן יש להוסיף לתוכנית את התנאי הבא:

if (pile > 21) {

System.out.println(“Please enter a number between 1 to 21”);

}

C#

if (pile > 21) {

Console.WriteLine(“Please enter a number between 1 to 21”);

}

חלקים מתוך התוכנית מוצגים להלן:

public static void main(String[] args) {

Scanner reader = new Scanner(System.in);

System.out.println("please enter a number");

int pile = reader.nextInt();

if (pile == 1) {

System.out.println("Take 1, but you lost :-( ");

}

if (pile == 2) {

System.out.println("Take 1");

}

if (pile == 3) {

System.out.println("Take 3");

}

//

//

המשיכו כך עד המקרה ה-21

//

//

if (pile == 21) {

System.out.println("Take one, hope the other player doesn't know how to win... ");

}

if (pile > 21) {

System.out.println(“Please enter a number from 1 to 21…”);

}

}

C#

static void Main(string[] args) {

Console.WriteLine("please enter a number");

int pile = int.Parse(Console.ReadLine());

if (pile == 1) {

Console.WriteLine("Take 1, but you lost :-( ");

}

if (pile == 2) {

Console.WriteLine("Take 1");

}

if (pile == 3) {

Console.WriteLine("Take 3");

}

//

//

המשיכו כך עד המקרה ה-21

//

//

if (pile == 21) {

Console.WriteLine("Take one, hope the other player doesn't know how to win... ");

}

if (pile > 21) {

Console.WriteLine(“Please enter a number from 1 to 21…”);

}

}

**הערה חשובה:**

ניתן לכתוב את מבנה התנאי בצורות נוספות, למשל כך:

if (pile == 3) {

System.out.println("Take 2 ");

}

C#

if (pile == 3) {

Console.WriteLine("Take 2 ");

}

שימו לב שהסוגריים המסולסלים שפותחים את הבלוק נמצאים בשורת התנאי. שיטה זו ברורה קצת פחות מבחינת הקשר בין הסוגר הפותח לזה שסוגר, אבל עדיין קל להבין את מבנה הבלוק.

או כך:

if (pile == 4) System.out.println("Take 3");

C#

if (pile == 4) Console.WriteLine("Take 3");

כאשר יש הוראה בודדת, ניתן לכתוב אותה ללא הסוגריים המסולסלים. אפשר לכתוב את ההוראה באותה שורה או לרדת שורה, בהתאם לנוחותכם.

 שימו לב

צורות כתיבה אלו לא מומלצות משתי סיבות. הראשונה היא שהן פוגעות בקריאות של קטעי התוכנית ולא שומרות על מבנה בלוק הפקודות. השנייה היא שאם נרצה להוסיף פקודות לבלוק בעתיד, יהיה עלינו להוסיף סוגריים מסולסלים המגדירים את הבלוק. לכן מומלץ להימנע מכך, שכן תלמידים רבים שוכחים להוסיף את הסוגרים ויוצרים בסופו של דבר תוכניות שגויות.



משימה 5: אפשרות שנייה

**פתחו** אלגוריתם המשתמש במבנה הבקרה של תנאי כך שכל תנאי ייכתב בהכללה לקבוצה של מספרים.

**כתבו** את האלגוריתם ראשית בשפה חופשית.

רמז: באילו מיליות קישור אתם משתמשים בשפה החופשית?

**פתרון:**

מחקירת משחק הנים במשימה 3 למדנו כי ניתן לחלק את המספרים השונים לקבוצות:

"המספרים הבעייתיים": 1, 5, 9, 13, 17, 21.

"המספרים שבהם כדאי לקחת 1": 2, 6, 10, 14, 18.

"המספרים שבהם כדאי לקחת 2": 3, 7, 11, 15, 19.

"המספרים שבהם כדאי לקחת 3": 4, 8, 12, 16, 20.

"המספרים הגדולים מ-21": לא קיימים במשחק.

תכנון אלגוריתם בשפה חופשית:

אם בערמה סוכריה אחת, הפסדת

אם בערמה 5 סוכריות או 9 סוכריות או 13 סוכריות או 17 סוכריות או 21 סוכריות, קח סוכריה אחת וקווה שהשחקן היריב לא יודע שהוא יכול לנצח. (אפשר לקחת גם 2 או 3, אך הבחירה במספר 1 יכולה לסייע בבלבול השחקן היריב).

אם בערמה 2 סוכריות או 6 סוכריות או 10 סוכריות או 14 סוכריות או 18 סוכריות, קח סוכריה אחת.

אם בערמה 3 סוכריות או 7 סוכריות או 11 סוכריות או 15 סוכריות או 19 סוכריות, קח 2 סוכריות.

אם בערמה 4 סוכריות או 8 סוכריות או 12 סוכריות או 16 סוכריות או 20 סוכריות, קח 3 סוכריות.

אם בערמה יותר מ-21 סוכריות, בקש מהמשתמש לכתוב מספר חדש בין 1 ל-21.

ייתכן שכתבתם את האלגוריתם כך:

אם ( (**ערמה שווה 2**) או (**ערמה שווה 6**) או (**ערמה שווה 10**) או (**ערמה שווה 14**) או (**ערמה שווה 18**) ) **הדפס "קח 1"**.

צורת כתיבה זו יוצרת תנאי שמורכב ממספר תנאים, כלומר תנאי מורכב. תנאי זה משתמש במילה מדוברת שאותה צריך לממש בשפת מחשב – המילה **או**.

### כתיבת תנאי מורכב

לצורך כתיבת תנאי מורכב עלינו להכיר את האופרטור הלוגי (מכונה גם "הקשר לוגי") **או**, הפועל על שני ערכים בוליאניים או על שני ביטויים בוליאניים. נעמיק בנושא האופרטורים הלוגיים בהמשך הפרק הזה, שכן אלו אופרטורים מיוחדים וחשובים במדעי המחשב.

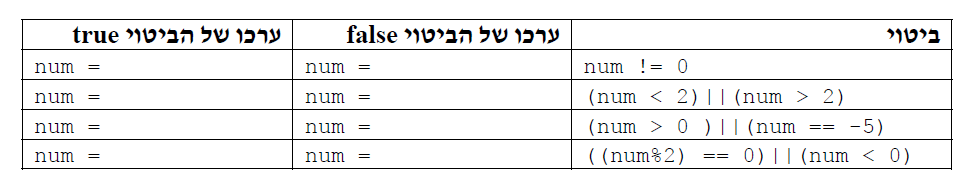
כפי שניתן להבין מהדוגמה, האופרטור **או** (**OR**) הוא אופרטור דו-מקומי הפועל על שני ביטויים בוליאניים. בשפה חופשית נגיד שאם לפחות ערך אחד מבין השניים הוא אמת, אז ערך הביטוי המורכב כולו הוא אמת. כלומר, מספיק שאחד מהתנאים המתאימים יתקיים (יש בערמה 2 או 6 או 10 או 14 או 18 סוכריות), והמשמעות היא שצריך לקחת סוכריה אחת.

האופרטור **או** (**OR**) מסומן ב-JAVA בסימון || (שני סימני | צמודים)

לפני המעבר למימוש הפתרון השני של משחק הנים, **תרגלו** את השימוש באופרטור:

  
משימה 6: תרגול האופרטור OR

**תנו** דוגמה לערך של המשתנה num שעבורו יהיה ערך הביטוי בעמודה הימנית true, ודוגמה אחרת שבה ערכו יהיה false.





משימה 7: מימוש האפשרות השנייה

כעת **ממשו** את האלגוריתם שכתבתם.

**פתרון:**

public static void main(String[] args) {

Scanner reader = new Scanner(System.in);

System.out.println("Please enter a number");

int pile = reader.nextInt();

if (pile == 1) {

System.out.println("Take 1, but you lost :-( ");

}

if (pile == 5 || pile == 9 || pile == 13 || pile == 17 || pile == 21) {

System.out.println("Take 1, hope the other player doesn't know how to win");

}

if (pile == 2 || pile == 6 || pile == 10 || pile == 14 || pile == 18) {

System.out.println("Take 1 ");

}

if (pile == 3 || pile == 7 || pile == 11 || pile == 15 || pile == 19) {

System.out.println("Take 2 ");

}

if (pile == 4 || pile == 8 || pile == 12 || pile == 16 || pile == 20) {

System.out.println("Take 3 ");

}

}

C#

static void Main(string[] args) {

Console.WriteLine("Please enter a number");

int pile = int.Parse(Console.ReadLine());

if (pile == 1) {

Console.WriteLine("Take 1, but you lost :-( ");

}

if (pile == 5 || pile == 9 || pile == 13 || pile == 17 || pile == 21) {

Console.WriteLine("Take 1, hope the other player doesn't know how to win");

}

if (pile == 2 || pile == 6 || pile == 10 || pile == 14 || pile == 18) {

Console.WriteLine("Take 1 ");

}

if (pile == 3 || pile == 7 || pile == 11 || pile == 15 || pile == 19) {

Console.WriteLine("Take 2 ");

}

if (pile == 4 || pile == 8 || pile == 12 || pile == 16 || pile == 20) {

Console.WriteLine("Take 3 ");

}

}

מהפתרון ניתן להבין כיצד בנוי תנאי מורכב. השתמשנו בסימן ||, שמשמעותו או ובאנגלית **OR**.



משימה 8: אפשרות שלישית

**פתחו** אלגוריתם המשתמש במבנה הבקרה של תנאי שיתאים לכל ערמה, לא משנה מה גודלה (גם אם הוא עולה על 21).

**כתבו** את האלגוריתם תחילה בשפה חופשית.

רמז: בדקו האם למספרים בחלוקה לקבוצות יש שארית חלוקה משותפת עבור מספר כלשהו.

**פתרון:**

נמצא את החוקיות על ידי ריכוז המידע וניסיון להכליל את הממצאים עליו. במשחק זה מתגלה קשר בין המספרים בכל קבוצה לבין שארית החלוקה במספר 4:

"המספרים הבעייתיים": 1, 5, 9, 13, 17, 21 – שארית החלוקה שלהם ב-4 היא 1.

"המספרים עבורם כדאי לקחת 1": 2, 6, 10, 14, 18 – שארית החלוקה שלהם ב-4 היא 2.

"המספרים עבורם כדאי לקחת 2": 3, 7, 11, 15, 19 – שארית החלוקה שלהם ב-4 היא 3.

"המספרים עבורם כדאי לקחת 3": 4, 8, 12, 16, 20 – שארית החלוקה שלהם ב-4 היא 0.

לפיכך, ניתן לבדוק את שארית החלוקה של מספר הסוכריות בערמה ב-4, במקום את מספר הסוכריות בערמה.

זהו צעד משמעותי מאוד בפתרון הבעיה, שכן החשיבה הזו מתנתקת מהדוגמה הנוכחית שבה יש לכל היותר 21 סוכריות, ומתמקד באלגוריתם שיפתור כל בעיה של נים עם כל גודל ערמה. הכלל מאפשר בעצם לחלק את כל המספרים בסך הכול ל-4 קבוצות.

כל האלגוריתם יתומצת לחמישה תנאים:

אם יש בערמה סוכריה אחת, קח אותה והפסדת

אם (שארית חלוקת מספר הסוכריות בערמה ב-4 היא 2), קח סוכריה אחת.

אם (שארית חלוקת מספר הסוכריות בערמה ב-4 היא 3), קח 2 סוכריות.

אם (שארית חלוקת מספר הסוכריות בערמה ב-4 היא 0), קח 3 סוכריות.

אם (שארית חלוקת מספר הסוכריות בערמה ב-4 היא 1) קח סוכריה אחת וייתכן שתפסיד.

האלגוריתם בשפה חופשית:

**אם** (מספר הסוכריות בערמה הוא 1), קח סוכריה אחת והפסדת,

**אחרת** **אם** (שארית חלוקת מספר הסוכריות בערמה ב-4 היא 2) קח סוכריה אחת,

**אחרת** **אם** (שארית חלוקת מספר הסוכריות בערמה ב-4 היא 3) קח 2 סוכריות,

**אחרת אם** (שארית חלוקת מספר הסוכריות בערמה ב-4 היא 0) קח 3 סוכריות,

**אחרת** קח סוכריה אחת וקווה שהשחקן היריב לא יודע שהוא יכול לנצח.

בפתרון זה ראינו צירוף מורכב מאוד של אם ואחרת...

כך נראה המימוש בסביבת העבודה:

public static void main(String[] args) {

Scanner reader = new Scanner(System.in);

System.out.println("Please enter a number");

int pile = reader.nextInt();

if (pile == 1) {

System.out.println("Take 1, but you lost");

} else // number is not 1

{

if (pile % 4 == 2) // but not 1

{

System.out.println("Take 1");

} else // no use of { } because the if statement is the only one

if (pile % 4 == 3) {

System.out.println("Take 2");

}

else

If(pile % 4 == 0) {

System.out.println("Take 3")

}

else // the remainder can be only 1

System.out.println("Take 1 and hope the other player doesn't know he can win ");

}

}

C#

static void Main(string[] args) {

Console.WriteLine("Please enter a number");

int pile = int.Parse(Console.ReadLine());

if (pile == 1) {

Console.WriteLine("Take 1, but you lost");

}   
 else // number is not 1

{

if (pile % 4 == 2) // but not 1

{

Console.WriteLine("Take 1");

} else // no use of { } because the if statement is the only one

if (pile % 4 == 3) {

Console.WriteLine("Take 2");

}

else

if(pile % 4 == 0) {

Console.WriteLine("Take 3")

}

else // the remainder can be only 1

Console.WriteLine("Take 1 and hope the other player doesn't know he can win ");

}

}

 שימו לב

הפעם התוכנה כבר לא מוגבלת ל-21 סוכריות, אלא מוכללת: כלומר, אפשר לשחק עם כל מספר סוכריות שייקלט.

המבנה המוצג כאן הוא שימוש בתנאי בתוך תנאי. שיטה זו נקראת **קינון תנאים** (nesting).

חשוב להדגיש את כללי השיוך של ה-else ל-if המתאים לו. ככלל, else תמיד משויך ל-if הקודם לו בתוכנית. אבל ישנן כמה מגבלות לכך:

ל-if זה לא משויך else קרוב יותר.

ה-if אינו כלול בתחום סוגריים מסולסלים אחר שמסתיים לפני ה-else.

חשוב לרשום את הוראות התנאי המקונן בצורה ברורה שתבהיר את השייכות של כל else.

את התנאי המקונן הכרנו כבר בפרק 1 במסגרת משחק המטבעות. **חזרו** למשחק המטבעות, בדקו מהו הפתרון שלו ופתרו את המשימה הבאה.



משימת אתגר 9: פתרון משחק נים בשורה אחת

**פתרו** את השאלה בשימוש בהוראה אחת בלבד.

עבור המספרים "הבעייתיים" **הדפיסו** "קח 0".

המימוש צריך להתאים לאלגוריתם הבא:

System.out.println("Enter your number");

int number = myScanner.nextInt();

int result = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

System.out.println("Take " + result + " candies");

C#

Console.WriteLine("Enter your number");

int number = int.Parse(Console.ReadLine());

int result = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

Console.WriteLine("Take " + result + " candies");

הפתרון האחרון שמצאנו באפשרות השלישית לאלגוריתם משחק הנים נראה פתרון יעיל למדי, במיוחד משום שהוא מתאים לפתרון בעיות שונות שבהן יש ערימות בגדלים שונים. אולם, אם תעמיקו עוד בבעיה, תוכלו למצוא אפילו הכללה משמעותית יותר: הכללה שיש בה קשר ישיר בין שארית החלוקה ב-4 לבין מספר הסוכריות שיש לקחת. במשחק שניתחנו עד כה, אפשר לקחת עד שלוש סוכריות – ולכן מחשבים את שארית החלוקה ב-4 ונותנים המלצות לפי הטבלה הבאה:

|  |  |  |
| --- | --- | --- |
| **שארית החלוקה ב 4** | **כמה כדאי לקחת** | **חישוב מתמטי, כמה כדאי לקחת** |
| 1 | מפסידים – אפשר לקחת 1 | שארית החלוקה ב-4 פחות 1 (אי אפשר לקחת 0) |
| 2 | 1 | שארית החלוקה ב-4 פחות 1 |
| 3 | 2 | שארית החלוקה ב-4 פחות 1 |
| 0 | 3 | שארית החלוקה ב-4 ועוד 3 |

טבלה זו נראית מקיפה ביותר, אך עדיין היינו רוצים לשלב את כל הנתונים בביטוי מתמטי אחד. כדי לבצע זאת נבדוק האם אפשר לשפר את ההכללה בפתרון האחרון במשימה – ההכללה לארבעה סוגי מספרים לפי השארית של החלוקה ב-4?

הניתוח האחרון מגלה כי בשלושה מתוך ארבעת המקרים, יש לקחת סוכריה אחת פחות משארית החלוקה ב-4. הביטוי המתמטי הבא מתאר הכללה זו: pile % 4 - 1.

אולם, כאשר שארית החלוקה ב-4 היא 0 (כמו במספרים 4, 8, 12 וכו'). ערך הביטוי המתמטי יהיה 1- (מינוס 1), ואנחנו רוצים שיודפס 3...

נרכז שוב את החקירה שלנו עד כה:

שארית 1 – צריך לקחת 0 (מספר מפסיד). הביטוי החשבוני number % 4 - 1 עובד.

שארית 2 – צריך לקחת 1. הביטוי החשבוני pile % 4 - 1 עובד.

שארית 3 – צריך לקחת 2. הביטוי החשבוני pile % 4 - 1 עובד.

שארית 0 – צריך לקחת 3. הביטוי החשבוני pile % 4 - 1 מחזיר מספר לא נכון ושלילי

**רגע חושבים!**

איזו פעולה מתמטית ניתן לבצע כך שנקבל תוצאה נכונה, כלומר 3 במקום 1-, אבל מבלי לפגוע בתוצאות של החישוב הקודם?

הוספה של הערך 4 למספר 1-, תציג את המספר שיש להדפיס – 3. הוספה זו תתאים במקרה של שארית 0. יש לכתוב את הביטוי שוב עם השינוי הנדרש לקבל את המספר 3:

result = 4 + pile % 4 - 1;

זהו פיתרון טוב למספרים שמתחלקים ב-4 ללא שארית אבל הוא אינו מתאים למקרים האחרים. יש לרכז שוב את הנתונים ולבצע בדיקה נוספת והפעם עם הביטוי החדש:

כשהשארית 1 – צריך לקחת 0, הביטוי החשבוני 4+ pile % 4 - 1 ייתן 4.

כשהשארית 2 – צריך לקחת 1, הביטוי החשבוני 4+ pile % 4 - 1 ייתן 5.

כשהשארית 3 – צריך לקחת 2, הביטוי החשבוני 4+ pile % 4 - 1 ייתן 6.

כשהשארית 0 – צריך לקחת 3, הביטוי החשבוני 4+ pile % 4 - 1 ייתן 3.

אלו הדפסות שגויות, אבל הביטו במספרים שהתקבלו הם רומזים על השלב האחרון שעלינו לבצע כדי להגיע לפתרון. נחפש פעולה שתעביר את המספרים שהתקבלו (4,5,6,3 למספרים הרצויים, 0,1,2,3...).

אתם מכירים פעולה כזו.

זוהי פעולה נוספת של שארית החלוקה ב-4: 4%4=0, 5%4=1, וכך הלאה. פעולה זו לא תשנה את התשובה הנכונה שהתקבלה במקרה של שארית 0 שכן 3%4=3.

כלומר התשובה היא:

result = (4 + (pile % 4 - 1)) % 4;

הדיון שהוצג כאן חשוב מאוד, שכן אחת מהמיומנויות החשובות במדעי המחשב היא היכולת לארגן את הנתונים ולכתוב תוכנית יעילה ומוכללת ככל שניתן.

**רגע חושבים!**

האם ניתן להשתמש בפתרון הזה למקרים מורכבים אפילו יותר, שבהם חוקי המשחק שונים ואפשר לקחת עד 5 סוכריות מהערמה?

### משתנה בוליאני

בפרק 2 הכרנו משתנים מטיפוסים שונים כגון int, double ו-String. בפרק זה נכיר משתנה חשוב נוסף – משתנה בוליאני. אומנם פגשנו עד כה בפרק את המונחים ערך בוליאני וביטוי בוליאני, אך חשוב לא להתבלבל ביניהם לבין משתנה בוליאני.

 הגדרה

משתנה בוליאני הוא משתנה אשר בו נשמר ערך בוליאני (כזכור ערך בוליאני יכול להיותtrue או false).

משתנה בוליאני מוגדר בדומה להגדרת משתנים אחרים – עם ערך התחלתי או בלעדיו.

boolean value;

boolean result = true;

**רגע חושבים!**

האם יצא לכם להיתקל בחיי היומיום ביישום של משתנים בוליאניים?

חשבו למשל על יומן הכיתה, שבו עמודת הנוכחות של כל תלמיד ותלמידה יכולה להיחשב משתנה בוליאני המקבל שני ערכים אפשריים: נוכח או לא נוכח. גם בטלפון הסלולרי יש משתנים בוליאניים: למשל, מצב הטיסה יכול להיות פעיל או כבוי בלבד, ולכן זהו משתנה בוליאני.

היזכרו בשאלת 21 הסוכריות. הוצג שם ביטוי בוליאני מורכב:

אם ( (**ערמה שווה 2**) או (**ערמה שווה 6**) או (**ערמה שווה 10**) או (**ערמה שווה 14**) או (**ערמה שווה 18**) ) **הדפס "קח 1"**.

והמימוש שלו היה:

if (pile == 2 || pile == 6 || pile == 10 || pile == 14 || pile == 18) {

System.out.println("Take 1 ");

}

C#

if (pile == 2 || pile == 6 || pile == 10 || pile == 14 || pile == 18) {

Console.WriteLine("Take 1 ");

}

ניתן להציב ערך של ביטוי בוליאני במשתנה בוליאני, ולאחר מכן לבדוק את ערך המשתנה. לדוגמה, התוכנית שהוצגה ניתנת למימוש כך:

boolean value = (pile == 2 || pile == 6 || pile == 10 || pile == 14 || pile == 18);

if (value) {

System.out.println("Take 1");

}

C#

bool value = (pile == 2 || pile == 6 || pile == 10 || pile == 14 || pile == 18);

if (value) {

Console.WriteLine("Take 1");

}

שימו לב שבמשתנה value יהיה רק אחד מהערכים true או false. בהתאם לכך יתבצעו הפעולות.

**רגע חושבים!**

האם אפשר לרשום את הבדיקה כפי שנכתבה או שצריך לרשום את הביטוי

if (value == true)?

שתי השיטות אפשריות, אולם אין צורך בהשוואה של משתנה בוליאני ל-true. משום שהערך המקורי של המשתנה value הוא גם הערך של הביטוי הבוליאני, למעשה הבדיקה מיותרת.

באופן מפורט יותר נסביר, שאם ערכו של value יהיה true, תוצאת ההשוואה value == true תיתן true, ואם ערכו של value יהיה false, תוצאת ההשוואה value == true תיתן false. המשמעות היא שתוצאת ההשוואה תהיה תמיד הערך של value, ולכן אין בעצם צורך להשוות בין הערך ל-true.

תוכלו להשתמש בכלי שנקרא טבלת אמת אשר עוזר לחשב ערך של ביטויים בוליאניים, ובמקרה זה:

|  |  |
| --- | --- |
| **value == true** | **value** |
| True | true |
| False | false |

ניתן לראות שההשוואה מיותרת שכן תוצאתה שווה תמיד לערכו המקורי של המשתנה.

נסכם את המושגים שהכרתם עד כה:

* **ערך בוליאני – ערך אמת או שקר, true או false.**
* **משתנה בוליאני** – משתנה אשר בו ניתן לשמור **ערך בוליאני**.
* **ביטוי בוליאני** (יכול להיות ערך בוליאני, משתנה בוליאני או תוצר של שימוש באופרטור מתאים כמו בדוגמאות שבטבלה במשימה 1).
* **אופרטור השוואה** – אופרטור המשווה שני ערכים כדוגמת מספרים או תווים.   
  האופרטורים שהוצגו הם בדיקת שיוויון (==), בדיקת אי שוויון (=!), בדיקה האם אחד הערכים גדול יותר מהשני (<), בדיקה האם אחד הערכים קטן יותר מהשני (>), בדיקה האם ערך אחד גדול או שווה לשני (=<), ובדיקה האם ערך אחד קטן או שווה לשני (=>).
* **אופרטור לוגי** – אופרטור שפועל על ביטוי בוליאני אחד או יותר ומחזיר תוצאה של ערך בוליאני. נעמיק בהיכרות איתם מייד בהמשך.
* **מבנה תנאי הבקרה if-else –** מבנה תנאי שפועל כך: **אם** ערכו של ביטוי בוליאני הוא אמת, נבצע בלוק פקודות, **אחרת** נבצע בלוק פקודות אחרות. לעיתים ניתן להשמיט את ה-else, כפי שהוסבר במשימה 8.

### אופרטורים לוגיים

כזכור, בפתרון משימה 5 הכרנו את האופרטור **או** (**OR**), שהוא אופרטור דו-מקומי (פועל על שני משתנים). במקום הניתוח המילולי שהוצג לפתרון בעיית הסוכריות, רצוי להשתמש בשיטה פורמלית להגדרת האופרטור – הצגת כל האפשרויות בטבלה.

ציינו כי אופרטור זה הוא אופרטור לוגי, כלומר אופרטור שפועל על ביטוי בוליאני אחד או יותר ותוצאתו היא ערך בוליאני. לאופרטור לוגי יש טבלאות אמת, אשר מציגות את כל התוצאות האפשריות שלו כאשר הוא פועל על שני ביטויים בוליאניים A ו-B.

בטבלה ניתן לזהות שערכו של הביטוי הבוליאני המורכב הוא true כאשר לפחות אחד מהביטויים הוא true.

|  |  |  |  |
| --- | --- | --- | --- |
| מה נאמר באינטראקציה | A || B | B | A |
| אם ערך הביטוי A הוא אמת וערך הביטוי B הוא אמת, אז ערך הביטוי A || B הוא אמת | true | true | true |
| גם אם רק אחד מבין שני הביטויים הוא אמת, למשל ערך הביטוי A הוא אמת וערך הביטוי B הוא שקר, אז ערך הביטוי A || B הוא אמת | true | true | false |
| או ההפך – אם ערך הביטוי A הוא שקר וערך הביטוי B הוא אמת, עדיין ערך הביטוי A || B הוא אמת | true | false | true |
| רק במקרה אחד ערך הביטוי כולו הוא שקר, וזאת כאשר שני הביטויים הם בעלי ערך שקר: אם ערך הביטוי A הוא שקר וערך הביטוי B הוא שקר, אז ערך הביטוי A || B הוא שקר | false | false | false |

כזכור האופרטור **או** (**OR**) מסומן ב-JAVA בסימון || (שני סימני | צמודים)

בפתרון השלישי לבעיית הסוכריות נעשה שימוש בתנאים מקוננים. תחילה נבדק את התנאי הראשון, ואם הוא מתקיים אז בלוק הפעולות ימשיך לבדיקת תנאי נוסף.

אם מתקיים **ביטוי בוליאני**, נבצע

אם מתקיים **ביטוי בוליאני**, נבצע

**סדרת פעולות**.

למעשה הדרישה היא ששני התנאים יתקיימו כדי שמשהו יתבצע. אפשר לכתוב את הדרישה גם בעזרת האופרטור **וגם** (**AND**). הדרישה תיראה כך:

אם מתקיים (**ביטוי בוליאני וגם** **ביטוי בוליאני**) נבצע **סדרת פעולות**.

האופרטור וגם (AND) מסומן בג'ווה בסימון && (שני סימני & צמודים).

גם את האופרטור הזה ניתן להגדיר בצורה פורמלית בעזרת טבלת אמת.

בהמשך נכיר תנאים מורכבים יותר, וטבלאות האמת ישמשו כלי נוח לשילוב בין תנאים רבים.

לפניכם טבלת האמת של האופרטור **וגם** (**AND**), הפועל על שני ביטויים בוליאניים. זהו ביטוי בוליאני מורכב הכולל שני ביטויים ואופרטור ביניהם.

בטבלה ניתן לזהות שערכו של הביטוי הבוליאני המורכב הוא true כאשר שני הביטויים הם true.

|  |  |  |  |
| --- | --- | --- | --- |
| מה נאמר באינטראקציה | A && B | B | A |
| רק במקרה אחד ערך הביטוי כולו יהיה אמת, וזאת כאשר ערך הביטוי A הוא אמת וערך הביטוי B הוא אמת, ואז ערך הביטוי A && B הוא אמת | true | true | true |
| אם רק אחד מבין שני הביטויים הוא אמת, למשל ערך הביטוי A הוא אמת וערך הביטוי B הוא שקר, אז ערך הביטוי A && B הוא שקר | false | false | true |
| וגם כאשר ערך הביטוי A הוא שקר וערך הביטוי B הוא אמת, עדיין ערך הביטוי A && B הוא שקר | false | true | false |
| כאשר שני הביטויים הם בעלי ערך שקר, כלומר ערך הביטוי A הוא שקר וערך הביטוי B הוא שקר, אז ערך הביטוי A && B הוא שקר | false | false | false |

  
משימה 3.5: תרגול האופרטור AND

פעילות אינטראקטיבית לבדיקת האופרטור **AND**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **(number3 > number2) && (number2 > number1)** | **(number2 > number1)** | **(number3 > number2)** | **number3** | **number2** | **number1** | **תרגיל** |
| true | true | true | 4 | 2 | 1 | 1 |
| false | true | false | 2 | 2 | 1 | 2 |
| false | false | true | 2 | 1 | 1 | 3 |
| false | false | false | 1 | 1 | 1 | 4 |
| true | true | true | 2 | 1 | 0 | 5 |

כדי לסכם את ההיכרות עם האופרטורים הלוגיים חשוב שנכיר אופרטור נוסף – ההיפוך הלוגי **NOT**. אופרטור זה פועל על ביטוי בוליאני בודד והופך את ערכו. כלומר, אם ערך הביטוי היה **אמת**, תוצאת האופרטור היא **שקר**, ואם ערכו היה **שקר** תוצאת האופרטור היא **אמת**. אופרטור זה מסומן בסימון **!** (סימן קריאה).

ובטבלה:

|  |  |
| --- | --- |
| **!a** | **a** |
| false | true |
| true | false |

  
משימה 7: אמת, שקר ומה שביניהם

**אמת, שקר ומה שביניהם**

**קבעו** את הערך של הביטויים המורכבים הבאים:

|  |  |  |
| --- | --- | --- |
| **תרגיל** | **הביטוי הבוליאני** | **ערך** |
| **1** | ישראל היא המדינה הקטנה בעולם **או** שהוותיקן היא המדינה הקטנה בעולם | true |
| **2** | ישראל היא המדינה הקטנה בעולם **וגם** הוותיקן היא המדינה הגדולה בעולם | false |
| **3** | ניצן הוא שם של בת **או** שם של בן | true |
| **4** | ניצן הוא שם של בת **וגם** שם של בן | true |
| **5** | חתול הוא יונק **וגם** כלב הוא זוחל | true |
| **6** | חתול הוא יונק **וגם** כלב הוא **לא** זוחל | true |
| **7** | חתול הוא יונק **או** כלב הוא **לא** זוחל | true |
| **8** | חתול הוא יונק **וגם** נחש הוא זוחל | true |
| **9** | המטבע נפל על הצד עץ **או** על הצד פלי | true |
| **10** | המטבע נפל על הצד עץ **וגם** על הצד פלי | false |

המשיכו לפעילות הבאה. המספרים לפעילות זו הם:

|  |  |  |  |
| --- | --- | --- | --- |
| **number4** | **number3** | **number2** | **number1** |
| 0 | 127 | 13 | 1 |

|  |  |  |
| --- | --- | --- |
| **ערך הביטוי** | **הביטוי הבוליאני** | **תרגיל** |
| true | (number3 > number2) && (number2 > number1) | 1 |
| false | !(number3 > number2) && (number2 > number1) | 2 |
| false | !(number3 > number4) || (number2 > number3) | 3 |
| true | (number2 % number3 == 13) || (number1<number4) | 4 |
| true | (number2 % number3 == 13) && (number1>number4) | 5 |
| true | (number4 % number3 == 13) || !(number1<number4) | 6 |
| true | (number2 / number3 == number4) || !(true) | 7 |
| true | !(number2 - number3 >= 0 || number1<number4) | 8 |
| false | !(number3 – number2 >= number1 \* number4) | 9 |
| false | !(number3 – number3 / number2 \* number2 > 0) | 10 |
| true | !(number3 – number3 % number2 \* number2 > 0) | 11 |

### העשרה: תנאים בלי מבנה הבקרה

**רגע חושבים!**

על אף שכותרת הפרק היא תנאים, האם ניתן לפתור שאלות מילוליות שמכילות תנאי רק באמצעות הכלים שלמדתם עד כה?

**פתרו** את המשימה הבאה מבלי להשתמש במבנה התנאי if… else… שלמדתם:



משימה 3.6: שאלת היפוך אבל בגישה אחרת

קלט: 0 או 1 (לא יהיה קלט אחר, אין צורך לבדוק זאת)

פלט: אם הקלט הוא 0 יודפס 1. אם הקלט הוא 1 יודפס 0.

רמז: מותר להשתמש בפעולות מתמטיות.

**השלבים לפתרון:**

הקלט יכול להיות רק 0 או 1, ואין צורך לבדוק אם יש קלט אחר.

האלגוריתם: האלגוריתם הטבעי הוא להשוות את הקלט למספר 0. אם הוא שווה לאפס, עליו להדפיס 1. אחרת עליו להדפיס 0. האלגוריתם אומנם עובד, אבל בהתאם להנחיה בשאלה, צריך לחפש פתרון שלא משתמש בתנאים. לפניכם אינטראקציה קצרה שיכולה לסייע במציאת הפתרון לשאלה ללא שימוש בתנאים:

דרך התרגיל למדנו כי לעיתים ניתן לממש תנאים גם בלי תבנית של תנאי, אבל במקרים רבים מימוש זה הוא מסורבל ואיננו אינטואיטיבי. עם זאת, חשוב להכיר אותו ולבדוק מימוש ללא תנאים כאחד מהפתרונות האפשריים בעתיד.



משימה מסכמת – התארגנות למבחן

קיבלתם מחשבון מדעי חדש וגיליתם שניתן לתכנת בו ב-JAVA, אולם הוא מוגבל ל-20 שורות קוד בלבד.

כהכנה למבחן במתמטיקה החלטתם לתכנת את הנוסחה של שורשי משוואה ריבועית. אומנם במבחן יהיה עליכם לבדוק את כל הדרך, אבל זו יכולה להיות שיטה לבדוק שצדקתם.

**הקלט:** הפרמטרים a, b ו-c.

**הפלט:** יוצג בצורה הבאה:

אם יש שני שורשים תודפס ההודעה שיש שני שורשים שונים, ויודפסו ערכי השורשים.

אם יש שורש בודד, תודפס הודעה שיש שורש בודד ויודפס ערכו.

אם אין שורשים, תודפס ההודעה "אין שורשים ממשיים".

במקרים מיוחדים כמו a = 0, תודפס הודעה מתאימה כמו "הפרמטרים אינם מייצגים פונקציה ריבועית".

דוגמאות שימושיות לבדיקת תקינות האלגוריתם:

לערכים a=1, b=0, c=-9 תודפס ההודעה: "למשוואה שני שורשים ממשיים, 3 ו 3-".

לערכים a=0, b=1, c=-9 תודפס ההודעה: "הפרמטרים אינם מייצגים פונקציה ריבועית".

### פעולות חיצוניות

כפי שראינו, תוכניות מחשב המבצעות משימות מורכבות בנויות למעשה ממקטעים שכל אחד מהם מבצע תת- משימה: למשל, הצב צייר משולש ולאחר מכן זז למיקום חדש וצייר משולש נוסף. ככל שהמשימה תהיה מורכבת יותר, כך חשוב יותר לחלק אותה לתתי-משימות.

החלוקה לתתי-משימות מאפשרת לחלק את המשימות בין מספר משתתפים. למשל אם כל תלמיד בכיתה היה אחראי על כתיבת אות אחת בעזרת הצב, ותצליחו למצוא שיטה נוחה לאחד בין האותיות, תוכלו לקלוט מהמשתמש את שמו ולכתוב אותו בצורה גרפית בעזרת הצב.

למשל כתיבת האות **ש** תיראה כך:

חלוקת העבודה בין מספר מתכנתים תקצר את זמן הפיתוח, אולם צורת העבודה שהכרנו עד כה אינה מתאימה לכתיבה של משימות מורכבות כאלו. איחוד קטעי הקוד ידרוש איסוף של שורות הקוד מכל המתכנתים ואיחוד שלהם בתוכנית אחת גדולה.

כדי להקל על השימוש בתתי-משימות ולחסוך בכתיבה אם יש משימות שמתבצעות כמה פעמים במהלך התוכנית, אפשר להגדיר פעולה שמבצעת תת-משימה מסוימת בלבד ולהשתמש בה כשיש צורך בביצוע תת- משימה זו. למשל, בתוכנית שבה יש צורך לצייר ריבועים רבים, במקום לכתוב את ההוראות לציור ריבוע שוב ושוב, אפשר להגדיר פעולה שמציירת ריבוע ולקרוא לה כמה פעמים. במצב זה נצטרך להגדיר נתונים מסוימים שקשורים לריבוע, כמו אורך הצלע, מיקום ההתחלה וכיוון הריבוע, אבל החיסכון יהיה ניכר.

הידעתם? בספרות המקצועית פעולה נקראת לעיתים פונקציה (function), שיטה (method), או פרוצדורה (procedure).

על מנת להבהיר את הרעיון המתואר לעיל, תוכלו לשחק במשחק המחשב [lightbot](http://lightbot.com/flash.html) (נסו לעבור את כל האתגרים של שלב 1 ולהגיע לשלב 2). הפעולות במשחק נקראות פרוצדורות (procedures).



משימה 3.4: תת-משימות שמתבצעות מספר פעמים

**פתחו** אלגוריתם שכותב מילים בצורה גרפית. **ממשו** את האלגוריתם בעזרת תוכנית.

למשל למילה שמש, התוצאה תראה כך:



מספר השורות הדרוש לכתיבת המילה הוא כ-80. אבל אם תוכלו לכתוב את האות ש בעזרת פעולה ולקרוא לה פעמיים, תחסכו כ-30 שורות קוד...

רמז: עליכם להסתייע בפעולות עזר מתאימות.

בצעו זאת במספר שלבים:

1. כתבו פעולה שמגדירה צב ומציירת את האות ש בגודל של לא יותר מ-20 X 20 צעדים, ושהפינה השמאלית התחתונה שלה במרכז המסך.
2. כתבו תוכנית ראשית שמשתמשת בפעולה שכתבתם בסעיף א' ומציירת את האות ש כשהפינה השמאלית התחתונה במרכז המסך.
3. שנו את הפעולה שכתבתם כך שהיא תצייר את האות כשהפינה השמאלית התחתונה שלה נמצאת במקום שייקבע בתוכנית הראשית ויישלח ממנה. הפעולה תקבל את המיקום ותזיז לשם את הצב לפני תחילת הציור.

**רגע חושבים!**

כיצד הפעולה תקבל את המיקום המחודש של הצב? בסוגיה זו נעסוק בהמשך.

1. קראו לפעולה שכתבתם מספר פעמים וציירו את האות ש במקומות שונים על המסך.
2. כתבו תוכנית ראשית שמגדירה צב, ושנו את הפעולה כך שהיא תקבל את הצב ואת המיקום ההתחלתי שבו יש לצייר את האות. התוכנית תצייר את האות בעזרת הצב, והפינה השמאלית התחתונה של האות תתחיל במקום שייקבע בתוכנית הראשית. בסיומה הצב יחזור לנקודת ההתחלה.
3. חזרו על השלבים לכתיבת האות מ.
4. שנו את התוכנית הראשית לכתיבת המילה שמש.

**שלב א:**

נתחיל מפעולה המוכרת לנו מפרקים קודמים שבהם ציירנו ציורים שונים. גם אותיות הן למעשה סוג של ציור.

**פתחו** אלגוריתם המצייר את האות ש באמצעות הצב ו**ממשו** אותו בסביבת העובדה.

**פתרון:**

התוכנית הבאה תצייר את האות ש בנקודת המיקום ההתחלתי של הצב:

public static void main(String[] args) {

Turtle zav1 = new Turtle();

zav1.tailUp();

zav1.moveForward(100);

zav1.tailDown();

zav1.moveForward(12);

zav1.moveBackward(12);

zav1.turnRight(90);

zav1.moveForward(4);

zav1.turnLeft(90);

zav1.moveForward(12);

zav1.moveBackward(12);

zav1.turnRight(90);

zav1.moveForward(4);

zav1.turnLeft(90);

zav1.moveForward(12);

zav1.moveBackward(12);

zav1.tailUp();

zav1.moveBackward(100);

}

C#

static void Main(string[] args) {

Turtle zav1 = new Turtle();

zav1.TailUp();

zav1.MoveForward(100);

zav1.TailDown();

zav1.MoveForward(12);

zav1.MoveBackward(12);

zav1.TurnRight(90);

zav1.MoveForward(4);

zav1.TurnLeft(90);

zav1.MoveForward(12);

zav1.MoveBackward(12);

zav1.TurnRight(90);

zav1.MoveForward(4);

zav1.TurnLeft(90);

zav1.MoveForward(12);

zav1.MoveBackward(12);

zav1.TailUp();

zav1.MoveBackward(100);

}

הצב שנוצר פנה בכיוון מעלה ולכן האות מצוירת בכיוון המתאים כדי שבסוף הצב לא יכסה את האות. הפעולה מקדמת אותו 100 צעדים קדימה ושם מציירת את האות, בסיום הפעולה מחזירה אותו 100 צעדים אחורה. בהמשך יהיה עליכם למקם את האות בכל מקום שתרצו על המסך, כך שעניין ההסתרה של האות על ידי הצב לא צריך להטריד אתכם.

לפניכם השלבים להעברת סדרת ההוראות מהתוכנית הראשית לפעולה, כך שניתן להשתמש בפעולה על ידי קריאה מהתוכנית הראשית.

הגדרת פעולה חיצונית:

public static void drawShin()

{

גוף הפעולה מכיל את ההוראות לביצוע

}

ניתן לראות שהגדרת הפעולה דומה להגדרת התוכנית הראשית. רצוי ששם הפעולה יהיה בעל משמעות ויבהיר את מטרתה, או לפחות ירמוז עליה.

חשוב להרחיב מעט על משמעות המילים בכותרת הפעולה. משמעות ההגדרה **public** היא שניתן לקרוא לפעולה זו מכל מקום במחלקה. משמעות ההגדרה **static** היא שהפעולה שייכת למחלקה. זאת בניגוד לפעולות אחרות שפגשנו, כמו moveBackWord, שהיו פעולות פנימיות של העצם שנוצר על ידי המחלקה צב (ראו פירוט נוסף בפרק 6).

המילה **void** היא מילה שמורה בשפה, ומשמעותה שבסיום הפעולה מוחזר הערך "כלום" למקום שאליו היא זומנה בתוכנית. לעומתה, הפעולה Math.max מחזירה ערך מטיפוס ממשי לאחר זימונה.

מכאן נשאר רק למלא את הפעולה בתוכן שכבר קיים בתוכנית הראשית והתוצאה תהיה:

את הפעולה המוכנה כותבים בתוך המחלקה מעל לתוכנית הראשית או מתחתיה (אין חשיבות לסדר הפעולות במחלקה).

כאשר נריץ את הקוד הכתוב במחלקה, התוכנית הראשית היא הראשונה שתרוץ. כדי שפעולה מסוימת שנכתוב תתבצע, יש לזמנה מתוך התוכנית הראשית או מתוך פעולה אחרת. פעולה מזומנת באמצעות כתיבת שמה בין סוגריים עגולים, כמתואר להלן.

public static void drawShin() {

Turtle zav1 = new Turtle();

zav1.tailUp();

zav1.moveForward(100);

zav1.tailDown();

zav1.moveForward(12);

zav1.moveBackward(12);

zav1.turnRight(90);

zav1.moveForward(4);

zav1.turnLeft(90);

zav1.moveForward(12);

zav1.moveBackward(12);

zav1.turnRight(90);

zav1.moveForward(4);

zav1.turnLeft(90);

zav1.moveForward(12);

zav1.moveBackward(12);

zav1.tailUp();

zav1.moveBackward(100);

}

C#

public static void DrawShin() {

Turtle zav1 = new Turtle();

zav1.TailUp();

zav1.MoveForward(100);

zav1.TailDown();

zav1.MoveForward(12);

zav1.MoveBackward(12);

zav1.TurnRight(90);

zav1.MoveForward(4);

zav1.TurnLeft(90);

zav1.MoveForward(12);

zav1.MoveBackward(12);

zav1.TurnRight(90);

zav1.MoveForward(4);

zav1.TurnLeft(90);

zav1.MoveForward(12);

zav1.MoveBackward(12);

zav1.TailUp();

zav1.MoveBackward(100);

}

יש לזמן את הפעולה מהתוכנית הראשית

public static void main(String[] args) {

drawShin();

}

C#

static void Main(string[] args) {

DrawShin();

}

זה הזמן לממש את הפעולה בפועל בסביבת העבודה. פתחו פרויקט חדש ו/או מחלקה חדשה, כתבו את הפעולה במקום המתאים, זמנו אותה והריצו את התוכנית בסביבת העבודה.

למעשה סיימתם את סעיפים א ו-ב.

בסעיף ג יש לקבוע את המיקום הרצוי שבה יתחיל ציור האות בתוכנית הראשית, ולשלוח מיקום זה לפעולה שבנינו. אתם כבר מנוסים בשליחת נתונים לפעולות: למשל, בהוראהMath.min(3,5) שלחתם את הנתונים 3 ו-5, ובהוראה zav1.moveForward(100); שלחתם את הנתון 100. וכך גם במקרה זה: אם תרצו למשל שהצב יתחיל לצייר את האות במרחק 60 צעדים מימין לנקודת ההתחלה שלו ובמרחק 30 צעדים קדימה ממנה, הרישום הדרוש הוא drawShin(60,30);.

### כותרת הפעולה

כותרת הפעולה מורכבת משם הפעולה ומרשימת פרמטרים שבעזרתם ניתן לשלוח אל הפעולה ערכים מתאימים. פרמטר הוא משתנה פנימי של הפעולה, המוכר רק בתוך הסוגריים המסולסלים שלה. כיוון שפרמטר זה אינו נגיש מחוץ לפעולה, הוא נקרא גם משתנה מקומי.

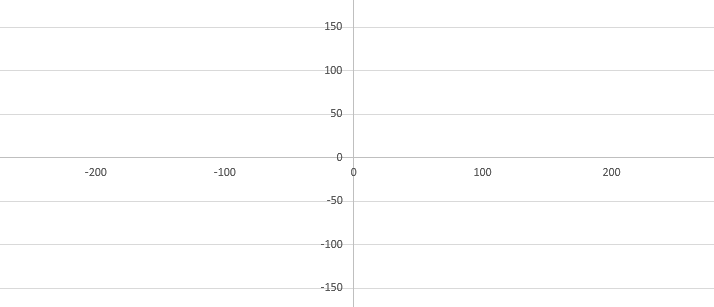
רשימת הפרמטרים בנויה מזוגות המכילים טיפוס ושם פרמטר, ומופרדים זה מזה בפסיקים. ישנה חשיבות רבה לטיפוסי הפרמטרים וסדר הגדרתם בכותרת הפעולה, כיוון שבעת זימון הפעולה יש לשלוח אליה את ערכי הפרמטרים בהתאם לסדר ולמספר הפרמטרים שהוגדרו.

לדוגמה, בפעולה הנדרשת, יש להגדיר שני פרמטרים מטיפוס int בשם x ו-y ובעת זימון הפעולה לשלוח אליהם את הערכים הרצויים:

public static void drawShin(int x, int y)

שמות המשתנים אליו יועברו הנתונים.

הסיבה שהם נקראים x ו y היא שיש כאן הצגה של מסך הצב כמערכת צירים.



לאחר כותרת הפעולה מופיע גוף הפעולה, הבנוי מסוגריים מסולסלים וביניהם רצף ההוראות הנכללות בו.

לחצו [כאן](https://drive.google.com/file/d/1esDl6hM58nGrDR1Jn_NkEHL7XMnPTgVK/view?usp=sharing) כדי לצפות בתוצאה.

והפתרון של סעיף ג' יראה כך:

public static void drawShin(int x, int y) {

Turtle zav1 = new Turtle();

//טענת כניסה: הפעולה מקבלת את המיקום של הפינה השמאלית התחתונה של האות

//ההנחה היא שהצב נמצא בראשית הצירים וכיוונו מעלה

//גודל האות מוגבל על ידי ריבוע של 20 על 20 צעדים

//טענת יציאה: הפעולה מציירת את האות ש', בסיום הציור הצב יוחזר למקומו ההתחלתי ובכיוונו ההתחלתי

// The method gets the location of the left corner of the letter.

// We assume the turtle is at the origin, heading upwards.

// A letters max dimensions are a 20X20 square.

// The method will draw the Hebrew letter Shin, at the end – the turtle will return to the origin heading upwards.

//הגעה לפינה השמאלית התחתונה של האות

// Moving to the left corner of the letter.

zav1.tailUp();

zav1.moveForward(y);

zav1.turnRight(90);

zav1.moveForward(x + 4);

zav1.turnLeft(90);

//ציור האות

// Drawing the letter

zav1.tailDown();

zav1.moveForward(12);

zav1.moveBackward(12);

zav1.turnRight(90);

zav1.moveForward(4);

zav1.turnLeft(90);

zav1.moveForward(12);

zav1.moveBackward(12);

zav1.turnRight(90);

zav1.moveForward(4);

zav1.turnLeft(90);

zav1.moveForward(12);

zav1.moveBackward(12);

//חזרה לנקודת המוצא

// Back to the origin

zav1.tailUp();

zav1.turnRight(90);

zav1.moveBackward(x + 20);

zav1.turnLeft(90);

zav1.moveBackward(y);

}

יש להקפיד כי בתחילת הפעולה יהיה תיעוד המסביר בדיוק מה היא עושה. זהו מעין חוזה הפעלה בין המתכנת שכתב את הפעולה לבין המתכנתים שישתמשו בה בעתיד. המתכנתים הבאים לא צריכים לדעת איך הפעולה פועלת למעשה, אלא רק להכיר את טענת הכניסה שלה (כולל ההנחות) ואת טענת היציאה.

כמו כן תוכלו להבחין בהערות בין שורות הפעולה שנועדו להסביר מה תפקידו של כל חלק בה.

**העשרה קצרה**:

עם זימון הפעולה, התוכנית פותחת סביבת עבודה חדשה שבה נכללים רק הפרמטרים שהוגדרו בכותרת הפונקציה, וכן משתנים מקומיים שהוגדרו בבלוק הפקודות של הפעולה (אם הוגדרו).

לכן, כאשר נזמן את הפעולה עם הערכים 60,30, ראשית ערכים אלו יושמו בתוך המשתנים לפי סדר הגדרתם, וטבלת המשתנים בסביבת העבודה החדשה תיראה כך:

|  |  |  |
| --- | --- | --- |
| **ערך** | **שם המשתנה** | **טיפוס המשתנה** |
| 60 | x | int |
| 30 | y | int |

לאחר שתתבצע ההוראה הראשונה של הפעולה, המגדירה צב חדש, טבלת המשתנים תיראה כך:

|  |  |  |
| --- | --- | --- |
| **ערך** | **שם המשתנה** | **טיפוס המשתנה** |
| 60 | x | int |
| 30 | y | int |
| הפניה | zav1 | Turtle |

הקריאה לפעולה בתוכנית הראשית תהיה:

public static void main(String[] args) {

drawShin(60, 30); // המספרים 60 ו 30 נבחרו רק לצורך הדוגמה, ניתן לבחור מספרים כרצונכם

}

נסו לפתור את סעיף ד בעצמכם (לזמן את הפעולה מספר פעמים ולצייר את האות ש במקומות שונים על המסך).

האם הפעולה תעבוד גם עם מספרים שליליים?



ד. יש צורך רק ברצף של זימונים. למשל הציור הזה מתבצע באמצעות ארבע זימונים:

drawShin(60, 30);

drawShin(150, 30);

drawShin(-60, 30);

drawShin(-150, 30);

ה. בסעיף זה התבקשתם לכתוב תוכנית ראשית שמגדירה צב.

אז איך שולחים צב לפעולה? ואיך מכינים את הפעולה לקבלת צב?

למעשה זה די דומה למה שעושים עם מספרים.

בתוכנית הראשית מגדירים צב ושלחים גם אותו לפעולה.

הזימון ייראה כך:

Turtle morla = new Turtle();

drawShin(morla, 60, 10);

כותרת הפעולה תיראה כך:

public static void drawShin(Turtle zav1, int x, int y)

בפעולה לא יהיה צורך להגדיר צב, שכן הוא כבר הוגדר בתוכנית הראשית. זה כל השינוי.

נרחיב לגבי שלוש נקודות:

1. שם הצב בתוכנית הראשית שונה מזה שבפעולה. זו לא בעיה. לצב מוקצה מקום בזיכרון המחשב, והתוכנית הראשית פונה לזיכרון זה בעזרת השם morla (צבה ענקית מהסרט "הסיפור שאינו נגמר"). הפעולה פונה לאותו מקום בזיכרון בשם zav1.
2. העברת פרמטרים מטיפוס פרימיטיבי כמו מספר שונה מהעברת פרמטרים מסוג עצם. כאשר אנחנו שולחים מספר, אנו שולחים את **הערך** שלו בלבד ואין קשר בין המקום בזיכרון המחשב בתוכנית הראשית לפעולה. שליחה מסוג זה נקראת העברה "by value", שכן מועבר הערך עצמו. כאשר אנו שולחים עצם, אנו שולחים למעשה הפניה לכתובת בזיכרון המחשב, וכך הפעולה והתוכנית הראשית עובדים על אותו מקום בזיכרון. שליחה זו נקראת "by address". לעובדה זו תהיה חשיבות בהמשך.
3. בהתאם למה שמתואר בהעשרה, המסקנה משתי הנקודות הראשונות היא הנקודה השלישית: העברה של ערך לא תשנה את ערך המשתנה המקורי בתוכנית שקראה לפעולה, והעברה של הפניה תשנה את הערך גם בתוכנית המקורית. ראינו זאת גם בטבלת המשתנים בסעיף ג בשאלה הקודמת.

סעיף ו. יש לחזור על השלבים לכתיבת האות **מ**.

הקושי העיקרי כאן יהיה לחשב את מספר הצעדים המדויק שדרוש לצב כדי לצייר את התנועה האלכסונית באות מ. אפשר להיעזר בחישוב טריגונומטרי כדי לדעת כמה צעדים נדרשים כדי לחזור לפינת האות ולאחר מכן לנקודת ההתחלה. אפשרות נוספת היא להנחות את הצב לחזור על צעדיו במדויק כך שיגיע חזרה לנקודת ההתחלה המדויקת. בפתרון שיוצג כאן נבחרה האפשרות השנייה, ומספר הצעדים נמצא בניסוי וטעייה.

הפתרון:

public static void drawMem(Turtle zav1, int x, int y) {

//טענת כניסה: הפעולה מקבלת צב ואת המיקום של הפינה השמאלית התחתונה של האות

//ההנחה היא שהצב נמצא בראשית הצירים וכיוונו מעלה

//גודל האות מוגבל על ידי ריבוע של 20 על 20 צעדים

//טענת יציאה: הפעולה מציירת את האות מ', בסיום הציור הצב יוחזר למקומו ההתחלתי ובכיוונו ההתחלתי

// The method gets a Turtle, the location of the left corner of the letter.

// We assume the turtle is at the origin, heading upwards.

// A letters max dimensions are a 20X20 square.

// The method will draw the Hebrew letter Mem, at the end – the turtle will return to the origin heading upwards.

//הגעה לפינה השמאלית התחתונה של האות

// Moving to the left corner of the letter.

zav1.tailUp();

zav1.moveForward(y);

zav1.turnRight(90);

zav1.moveForward(x + 4);

//ציור האות

// Drawing the letter

zav1.tailDown();

zav1.turnLeft(90);

zav1.moveForward(12);

zav1.moveBackward(5);

zav1.turnRight(45);

zav1.moveForward(8);

zav1.turnRight(45);

zav1.moveForward(4);

zav1.turnLeft(90);

zav1.moveBackward(12);

zav1.turnRight(90);

zav1.moveBackward(4);

zav1.moveForward(4);

zav1.turnRight(90);

zav1.moveBackward(12);

zav1.turnLeft(90);

zav1.moveBackward(4);

zav1.turnLeft(45);

zav1.moveBackward(8);

zav1.turnLeft(45);

zav1.moveBackward(7);

zav1.turnRight(90);

//חזרה לנקודת המוצא

// Back to the origin

zav1.tailUp();

zav1.moveForward(16);

zav1.moveBackward(x + 20);

zav1.turnLeft(90);

zav1.moveBackward(y);

}

ז. נותר לשנות את התוכנית הראשית לכתיבת המילה שמש.

כעת ניתן להבין עד כמה זה קל. כל מה שעלינו לכתוב בתוכנית הראשית היא ההוראה להגדרת צב, ולאחר מכן את זימוני הפעולות הרצויות עם הפרמטרים המתאימים:

נסכם מה ניתן ללמוד מהשימוש בפעולות.

המובן מאליו הוא שהחיסכון בכתיבה הולך ונהיה משמעותי כאשר אנו משתמשים במשפטים ארוכים, שכן האותיות החוזרות על עצמן מתרבות. כך למשל במשפט "שימי, שמת שום ומשמש בשומשום שבשמש?" יש 35 תווים (כולל רווחים וסימני פיסוק), אבל בגלל שהאות ש חוזרת 10 פעמים והאות מ חוזרת 6 פעמים, בחזרה על פעולת הציור של אותיות אלה נוכל לחסוך מעל 300 שורות קוד.

Turtle morla = new Turtle();

drawShin(morla, 60, 10);

drawMem(morla, 40, 10);

drawShin(morla, 20, 10);

לצד החיסכון, כך נוכל גם לשפר משמעותית את יכולתם של מתכנתים אחרים להבין את הקוד שכתבנו. למשל בזימון הפעולה drawShin יהיה ברור הרבה יותר מה עושה התוכנית, מאשר בניסיון להבין רצף פעולות של הרמת זנב, התקדמות ופנייה. תכונה זו נקראת **קריאות.**

רווח נוסף יהיה היכולת לשנות את הקוד בקלות. לדוגמה, אם לאחר ציור ההודעה הגרפית נחליט שצריך להאריך קצת את הקו של האות יו"ד, במקום לעדכן את ההוראות בכל המקומות שבהם כתבנו הוראת הדפסה עבורה, תצטרכו לעדכן את ההוראות רק במקום אחד – בפעולה. התוספת המשמעותית ביותר היא שכך תוכלו לחלוק את הקוד שלכם עם כל העולם. יכולות אלו יתרמו ל**תחזוקה ולשיפור** של התוכנית.

### פעולה חיצונית שמחזירה ערך

ראינו בעבר פעולות שקיבלו אפס פרמטרים או יותר, ומשימתן הייתה להחזיר ערך אחר – לדוגמה, הפעולה מהספרייה המתמטית להגרלת מספר אקראי או לחישוב מספר מקסימלי מבין שני מספרים.

ראינו גם פעולות שהחזירו את הערך המיוחד void, שבעצם לא מחזיר דבר.

נראה כיצד בנויה פעולה שמחזירה ערך.

שם המשתנה אליו יועבר הפרמטר

טיפוס הערך המוחזר

שם הפעולה

public static double square(int number)

ההגדרה בכותרת מאפשרת לקבל ערך מהטיפוס המתאים (במקרה זה int) אל תוך המשתנה number. שם הטיפוס שלפני שם הפעולה בכותרת מעיד שהפעולה מחזירה ערך מטיפוס זה (במקרה זה double).

למשל, פעולה זו מחזירה את ריבוע המספר שהתקבל כפרמטר:

public static int square(int number) {

return number \* number;

}

ההוראה **return** היא ההוראה האחרונה שתתבצע בפעולה. היא מחזירה ערך לפעולה שקראה לה ומסיימת את ביצוע הפעולה.



**כתבו** פעולה שמקבלת **שלושה פרמטרים מטיפוס שלם** **ומחזירה ערך בוליאני**.

הפעולה תחזיר true אם הערכים שהתקבלו יכולים לייצג צלעות של משולש,

ואם אי אפשר להרכיב משולש מצלעות באורכים אלו הפעולה תחזיר false.

לפני שתתחילו לבדוק מה נדרש במשימה, הגדירו את המבנה והכותרת של הפעולה המבוקשת. כותרת הפעולה שמקבלת שלושה ערכים מטיפוס שלם ומחזירה ערך בוליאני תראה כך:

public static boolean isTriangle(int side1, int side2, int side3)

C#

public static bool IsTriangle(int side1, int side2, int side3)

שם הפעולה מבהיר מה המשימה המתבצעת.

ועכשיו נסו להבין מהן הבדיקות שיש לבצע כדי לדעת האם הערכים שהתקבלו מתאימים ליצירת משולש.

בשאלות גאומטריות, במקרים רבים רצוי להתחיל בציורים – נסו את הצלעות 6, 6, 3. זה יהיה משולש שווה שוקיים... לא נראה שיש פה בעיה.

נסו לבדוק משהו קיצוני יותר – 6, 1, 1... כאן יש בעיה: שתיים מהצלעות קצרות מכדי שנוכל ליצור משולש עם הצלע הארוכה.

המשיכו לצייר ונסו שילובים שונים – למשל 4,4,8, או 3,8,4, ואולי 0, 0 ו 0... האם תוכלו לנסח מסקנה כוללת?

במחקר שערכנו, אפשר להגיע למסקנה שיש כלל ברור לבדיקה.

(בעמוד הבא)

כלל זה נקרא **אי-שוויון המשולש**. התנאי ההכרחי והמספיק ליצירת משולש משלוש צלעות הוא שסכום האורכים של כל שתי צלעות יהיה גדול מהצלע השלישית.

איך תכניסו את התנאי הזה לפעולה?

public static boolean isTriangle(int side1, int side2, int side3) {

if (side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 > side1) return true;

else return false;

}

C#

public static bool IsTriangle(int side1, int side2, int side3) {

if (side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 > side1) return true;

else return false;

}

שתי הערות לפעולה מסוג זה:

1. לאור העובדה שההוראה return היא האחרונה להתבצע ניתן היה לרשום את הפעולה גם כך:

public static boolean isTriangle(int side1, int side2, int side3) {

if (side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 > side1) return true;

return false;

}

C#

public static bool IsTriangle(int side1, int side2, int side3) {

if (side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 > side1) return true;

return false;

}

אם התנאי לא היה מתקיים, ההוראה return true לא הייתה מתבצעת והתוכנית הייתה ממשיכה לשורה הבאה. בשורה הזו יש הוראה יחידה ללא בדיקה של החזרת הערך false. בגלל שאנחנו יודעים שכיסינו את כל אפשרויות ההחזרה, הוראה זו תתבצע רק אם נגיע אליה, ולכן השימוש ב-else אינו הכרחי במקרה זה.

1. כיוון שהערך שמוחזר הוא בוליאני, ניתן לרשום את הפעולה גם כך:

public static boolean isTriangle(int side1, int side2, int side3) {

return (side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 > side1);

}

C#

public static bool IsTriangle(int side1, int side2, int side3) {

return (side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 > side1);

}

אם התנאי מתקיים אזי ערכו הוא true, ולכן יוחזר הערך true. אם הוא לא מתקיים אזי ערכו הוא false, ולכן יוחזר הערך false. למעשה בכל מקרה מוחזר הערך הבוליאני של התנאי ולכן אין צורך בהשוואה.

##### סיכום פרק תנאים ומחלקות עזר

* **ערך בוליאני** הוא אחד מהערכים **אמת** או **שקר**,ובסביבת העבודה **true** או **false**.
* **אופרטור לוגי** הוא פעולה הפועלת על **ביטוי בוליאני** אחד או יותר, ותוצאתו היא ערך בוליאני.
* **האופרטור הלוגי OR** פועל על שני אופרנדים ומחזיר ערך בוליאני. הערך **אמת** יוחזר אם לפחות אחד מהערכים של הביטויים הבוליאניים שעליהם הוא פועל הוא **אמת**.
* **האופרטור הלוגי AND** פועל על שני אופרנדים ומחזיר ערך בוליאני. הערך **אמת** יוחזר רק אם שני הערכים של הביטויים הבוליאניים שעליהם הוא פועל הם **אמת**.
* **האופרטור הלוגי NOT** פועל על אופרנד יחיד ומחזיר ערך בוליאני. הערך שיוחזר יהיה הפוך מערך הביטוי הבוליאני שעליו הוא פועל: לערך **אמת** יוחזר **שקר**, ולערך **שקר** יוחזר **אמת**.
* **משתנה בוליאני** הוא משתנה אשר בו נשמור **ערך בוליאני**.
* **ביטוי בוליאני** הוא ערך בוליאני או משתנה בוליאני או הפעלת אופרטור השוואה או הפעלת אופרטור לוגי על ערכים בוליאניים, משתנים בוליאניים או ביטויים בוליאניים. הערך של ביטוי כזה הוא או **אמת** או **שקר**, ובסביבת העבודה **true** או **false**.
* **ביצוע מותנה** הוא **מבנה בקרה** שבו המחשב מבצע סדרת פעולות על פי ערכו של **ביטוי בוליאני**.
* תבנית **מבנה הבקרה תנאי**:
* אם מתקיים תנאי מסוים, נבצע בלוק פקודות.
* אחרת נבצע בלוק פקודות (כנראה אחרות).

או בכתיבה אחרת:

* אם ערך ביטוי בוליאני הוא אמת, נבצע בלוק פקודות.
* אחרת נבצע בלוק פקודות (כנראה אחרות).
* **טבלת אמת** היא טבלה אשר מציגה את כל הערכים האפשריים לתוצאת חישוב ביטוי בוליאני בהתאם לכל הערכים האפשריים של מרכיביו.
* **קינון תנאים (nesting)** – תנאי מקונן הוא תנאי אשר מופיע בסדרת פעולות מותנית.
* **פעולה חיצונית** (נקראת גם פונקציה – function, שיטה –method , ופרוצדורה – procedure) – פעולה ששייכת למחלקה ואינה שייכת לעצם. בפרק 6 נרחיב על עצמים ועל פעולות פנימיות שהן הפעולות ששייכות לעצמים.
* **public** – מאפיין הרשאה לשימוש בפעולה. משמעותו היא שניתן לזמן את הפעולה מכל מקום בתוכנית. בהמשך נכיר רמות הרשאה נוספות.
* **static** –מציינת שפעולה חיצונית שייכת למחלקה ואין צורך להגדיר עצם כדי להשתמש בה.
* **void** היא מילה שמורה בשפה שאומרת כי הפעולה מחזירה את הערך "כלום".

חמישה בחמש

לפניכם רשימת נושאים לעבודת חקר קצרה.  
בחרו את אחד מהנושאים ברשימה, והכינו מצגת בת חמישה שקפים.  
לרשותכם חמש דקות להציג בפני הכיתה את הנושא הנבחר.

1. בפרק הוצג מסך הצב כמערכת צירים – כיצד זה קשור לאופן שבו עובד עכבר המחשב שלכם?
2. ייתכן שנתקלתם בעבר בעובדה שבניגוד לצב שתנועתו מוגדרת במספר צעדים, נקודות הצבע שעל המסך מוגדרות בפיקסלים. מהו פיקסל ומהי רזולוציה? כיצד הנושא קשור למסכי LCD?
3. השתמשנו במונח מבנה בקרה. ננצל את ההזדמנות ללמוד קצת על מונח מאותה משפחה, אחד מאבות המשפחה בעל השם המוזר "קוד ספגטי".
4. ניתוח לוגי נושק לעיתים לניתוח הסתברותי. הציגו את בעיית מונטי הול (The Monty Hall Problem), בעיה מעניינת במיוחד. מומלץ לצפות במספר סרטונים לפני שמחליטים כיצד להציג את הבעיה ופתרונה. כדאי להתחיל בסרטון ה-YOUTUBE [The Monty Hall Problem](file:///C:\Users\user\Desktop\ספר%20מדעי%20המחשב\פרק%203\The%20Monty%20Hall%20Problem).
5. דוגמה לקבלת החלטות ברמה מתקדמת במכונה אפשר ללמוד מווטסון של IBM, אשר ניצח במשחק מול מתחרים אנושיים. צפו בסרטון ה-YOUTUBE בקישור [IBM's Watson Supercomputer Jeopardy](https://www.youtube.com/watch?v=WFR3lOm_xhE), ושימו לב לאחוזי הוודאות שהמחשב מייחס לתשובות האפשריות שהוא מצא.
6. ראינו שניתוח מתמטי שמקדים לכתיבת תוכנית יכול להיות מועיל למדי. צאו קצת מעולם התכנות וחזרו לחטיבת הביניים, שם גיליתם ש-a0 הוא 1 (אם לא עשיתם זאת, נסו להבין למה בעזרת רשת האינטרנט), וש-0 בכל חזקה הוא 0 – כלומר ש 0a הוא 0. קחו השאלה שלב אחד הלאה ונסו לבדוק כמה זה 00. אפשר להיעזר בסרטון ה-YOUTUBE [what is 0 to the power of 0](file:///C:\Users\user\Desktop\ספר%20מדעי%20המחשב\פרק%203\what%20is%200%20to%20the%20power%20of%200).
7. ראינו שימוש בתנאים לקבלת החלטות, אבל האם תוכלו לנבא אם ג'ון ישחק טניס בהתאם ליום השבוע? מעניין ללמוד קצת על קבלת החלטות ברמת ניבוי, כיצד היא קשורה לתוכנות איסוף מידע, והיכן פועלות התוכנות הללו? למדו אותנו על עצי החלטה... צפו בהרצאה ראשונה [decision tree 1 john tennis](file:///C:\Users\user\Desktop\ספר%20מדעי%20המחשב\פרק%203\decision%20tree%201%20john%20tennis).
8. המונח מיקור חוץ (outsourcing)הוא שיטת ניהול שבה מוציאים מהארגון את כל הפעולות שאין הכר לבצוע בתוכו. קראו קצת על הרעיון והשוו אותו לשימוש בפעולות חיצוניות.
9. בחנו את חידת הכובעים בגרסה הפשוטה (חמישה כובעים ושלושה אנשים). תוכלו לראות סרטון מעניין הכולל את התשובה [puzzle 5 hats 3 men](file:///C:\Users\user\Desktop\ספר%20מדעי%20המחשב\פרק%203\puzzle%205%20hats%203%20men). חפשו חידות לוגיות דומות ונסו לרכז את הנתונים בטבלת מעקב.

#### פרק 3:ביצוע חוזר

#### הקדמה

עד כה הכרנו בעיות שנפתרות באמצעות אלגוריתם של שלבים סדרתיים. כל שלב באלגוריתם בוצע פעם אחת בלבד, ואם זו משימה שביצועה תלוי בתנאי – ייתכן שלא בוצע כלל. אולם יש בעיות שכדי לפתור אותן יש לבצע את אותו שלב או הוראה יותר מפעם אחת, ואולי אף מספר רב של פעמים. בפרק זה נכיר אלגוריתמים אשר מורים על ביצוע חוזר ונשנה של אותה הוראה או משימה.



משימה 1: ושוב עולה המנגינה

#### חלק א'

**כתבו** אלגוריתם המנגן מנגינה באורגן וחוזר עליה חמש פעמים. תוכלו להיזכר בתהליך הייבוא של מחלקת האורגן כפי שעשינו בפרק 1 (קישור).

**פתרון:**

בשלב הראשון יוצרים אורגן חדש, ומקליטים באמצעותו מנגינה חדשה, למשל: דו, רה, מי, דו, מי, דו, מי, פה.

משמיעים באמצעות האורגן את המנגינה חמש פעמים.

מכיוון שידוע לנו כיצד ליצור אורגן, להקליט מנגינה ולהשמיע אותה, אנו יכולים לכתוב את ההוראה להשמעת ההקלטה חמש פעמים בזו אחר זו.

**רגע חושבים!** מה תעשו אם תתבקשו לחזור על ההקלטה מאה פעמים? האם תכתבו מאה פעמים את ההוראה לאורגן להשמיע את ההקלטה?

כדי להורות לאורגן לבצע אותה משימה מספר פעמים, עליכם להכיר מבנה אלגוריתמי חדש: **הוראה לביצוע-חוזר**, או הוראת חזרה.

הוראה לביצוע-חוזר נקראת גם לולאה (loop), וקבוצת ההוראות לביצוע הכלולות בה נקראת "גוף הלולאה". בדומה לכתיבת הוראת תנאי, גם בכתיבת לולאה יש להקפיד על הזחה (indentation) מתאימה, וקבוצת ההוראות שיש לחזור על ביצוען מוזחת פנימה.

אנו נשתמש כאן בהוראה המיושמת ב java- במשפט for, ומכאן ואילך נכנה אותה בקיצור **לולאת for** או **משפט for**.

בבעיה שאנו מנסים לפתור, ניעזר במשתנה שיסייע לנו למנות כמה פעמים אנו חוזרים על סדרת הוראות מוגדרת. משתנה זה נקרא משתנה בקרה. בהוראה שלפנינו משתנה הבקרה משמש כמונה, ואנו נקרא לו בשםcounter .

אם ברצוננו להשמיע את הקלטת האורגן חמש פעמים, נכתוב משפט for בצורה הבאה:

for (int counter = 1; counter<= 5; counter++) {

ההוראות לביצוע

}

באופן כללי, במשפט for מוזן ערך התחלתי במשתנה הבקרה לפי הרכיב הראשון במשפט. לאחר מכן מתבצעת בדיקת התנאי המתואר ברכיב השני. אם ערכו של התנאי הוא true, ההוראות מתבצעות. בתום ביצוע קבוצת ההוראות גדל ערכו של משתנה הבקרה לפי הרכיב השלישי, ושוב מתבצעת בדיקת התנאי. אם ערכו של התנאי הוא true מתבצעת קבוצת ההוראות הבאה, וכן הלאה עד שערכו של התנאי יהיה false – ואז מסתיים הביצוע.

 שימו לב

ההוראה counter++ היא למעשה הוראת השמה מקוצרת, השקולה להוראת ההשמהcounter = counter+1 .

אפשר להשתמש בהוראה זו בכל מקום בתוכנית, לא רק במשפטfor .

**רגע חושבים!**

מה יבצע המחשב במשפט for הבא?

for (counter = 0; counter j<5; counter++)

{

ההוראות לביצוע האורגן מנגן את ההקלטה

}

גם במקרה זה ההוראות לביצוע מתבצעות חמש פעמים: פעם אחת כאשר ערכו של counter שווה ל-0, פעם שנייה כאשר ערכו שווה ל-1, ובפעם החמישית והאחרונה כאשר ערכו שווה ל-4. כאשר ערכו של counter גדל שוב ומגיע ל-5, התנאי להמשך הביצוע כבר לא מתקיים והביצוע החוזר מסתיים.



משימה 1: ושוב עולה המנגינה

#### חלק ב'

**ממשו** את האלגוריתם שהתבקשתם לכתוב במשימה 1: **כתבו** תוכנית שבה האורגן מנגן חמש פעמים את המנגינה שהקלטתם.

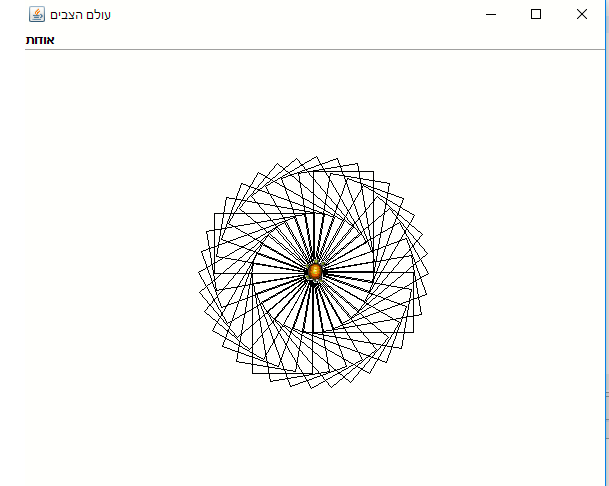


משימה 2: הצב שומר המסך

#### חלק א'

**כתבו** אלגוריתם המצייר באמצעות הצב שהכרנו 36 מלבנים היוצאים ממרכז עולמו של הצב, כך שכל מלבן מתחיל 10 מעלות ימינה מהמלבן שקדם לו.

**ראו** את ההדגמה הבאה:



נפרק את האלגוריתם לשתי משימות פשוטות:

**א. הגדרת המשתנים ואתחולם:**

על מנת לצייר נגדיר צב חדש. נוריד את זנבו של הצב למטה כדי שיוכל לצייר.

**ב. הצב מצייר 36 מלבנים:**

נשים לב כי אורך המלבן הוא 90 צעדי צב ורוחבו 50.

נשתמש במבנה החזרה שלמדנו.

יש לחזור 36 פעמים על ההוראות הבאות:

**הצב מצייר מלבן 90X50, ואז פונה ימינה בזווית של 10 מעלות**

**ממשו** את האלגוריתם – כתבו את התוכנית

התוכנית המתאימה מוסתרת.

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

{

Turtle tr1 = **new** Turtle();

tr1.tailDown();

tr1.setDelay(100);

**for**(**int** k=1 ;k<=36; k++)

{

//draw rectangle

tr1.moveForward(90);

tr1.turnRight(90);

tr1.moveForward(50);

tr1.turnRight(90);

tr1.moveForward(90);

tr1.turnRight(90);

tr1.moveForward(50);

tr1.turnRight(90);

tr1.turnRight(10);

}

}

C#

**public** **static** **void** Main(דtring[] args)

{

Turtle tr1 = **new** Turtle();

tr1.TailDown();

tr1.SetDelay(100);

**for**(**int** k=1 ;k<=36; k++)

{

//draw rectangle

tr1.MoveForward(90);

tr1.TurnRight(90);

tr1.MoveForward(50);

tr1.TurnRight(90);

tr1.MoveForward(90);

tr1.TurnRight(90);

tr1.MoveForward(50);

tr1.TurnRight(90);

tr1.TurnRight(10);

}

}

**רגע חושבים!**

אנו רוצים שהצב יצייר מספר מלבנים לבחירת המשתמש – לדוגמה, המשתמש יקליד 72, והצב יצייר 72 מלבנים. כיצד זה אפשרי?



משימה 2: הצב שומר המסך

#### חלק ב'

**כתבו** אלגוריתם חדש למטלה זו. רמז: האם צריך לתכנן מחדש אלגוריתם לביצוע מטלה זו? או אולי רק לשנות מעט את האלגוריתם שכבר נכתב?

**שנו** את האלגוריתם **וכתבו** תוכנית שתממש את האלגוריתם לאחר השינוי.

הערה: במקרה זה ייתכן יותר מפתרון אחד

אפשר להפנות את הצב ב-10 מעלות ימינה, ואפשר להחליט שכעת הפנייה מתבצעת ב-5 מעלות בלבד



משימה 3 – הצב מצייר שומר מסך צבעוני

#### חלק א'

את החלק הראשון במשימה עליכם לפתור לבד.

**קלט:** האלגוריתם מקבל צב ואורך צלע.

**פלט:** האלגוריתם מצייר באמצעות הצב ריבוע שאורך צלעו X.

הצב חוזר ארבע פעמים על ההוראות הבאות:

הצב צועד קדימה X צעדים ואחר כך מסתובב ב-90 מעלות ימינה

ממשו את הפעולה בסביבת העבודה.

לאחר שפתרתם לבד את החלק הראשון, ניגש כעת לחלק השני במשימה ביחד.

#### חלק ב'

**כתבו** אלגוריתם המצייר בעזרת הצב 36 ריבועים, כך שכל ריבוע זז ב-10 מעלות ימינה לעומת הריבוע הקודם. הריבועים צריכים להיות בשלושה צבעים לסירוגין: אדום, ירוק, כחול.

נתחיל בפירוק הבעיה לתת-משימות.

א. הגדרת המשתנים ואתחולם.

ב. הצב מצייר 36 ריבועים. כדי לצייר ריבוע נזמן את הפעולה שכתבנו במשימה 3.

לפני ציור הריבוע נבדוק מה שארית החלוקה בין מספר הריבוע לשלוש:

* עבור שארית 0 הצב יצייר בצבע אדום.
* עבור שארית 1 הצב יצייר בצבע ירוק.
* עבור שארית 2 הצב יצייר בצבע כחול.

**התוכנית עבור משימה 4**

**import** unit4.turtleLib.Turtle;

**public** **class** Ex4

{

**public** **static** **void** drow\_square(Turtle dr\_tr1 , **int** x )

{

**for**(**int** k =1; k<=4 ; k++)

{

dr\_tr1.moveForward(x);

dr\_tr1.turnRight(90);

}

}

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

{

Turtle tr1 = **new** Turtle();

tr1.tailDown();

tr1.setDelay(100);

**for**(**int** k=1 ;k<=36; k++)

{

**if**(k%3==0)

tr1.setTailColor(Color.red);

**else**

**if**(k%3==1)

tr1.setTailColor(Color.green);

**else**

tr1.setTailColor(Color.blue);

drow\_square(tr1, 100);

tr1.turnRight(10);

}

}

}

C#

using Unit4.TurtleLib;

using System.Drawing;

**class** Ex4

{

**static** **void** Draw\_square(Turtle dr\_tr1 , **int** x )

{

**for**(**int** k =1; k<=4 ; k++)

{

dr\_tr1.MoveForward(x);

dr\_tr1.TurnRight(90);

}

}

**static** **void** Main(string[] args)

{

Turtle tr1 = **new** Turtle();

tr1.TailDown();

tr1.SetDelay(100);

**for**(**int** k=1 ;k<=36; k++)

{

**if**(k%3==0)

tr1.SetTailColor(Color.red);

**else**

**if**(k%3==1)

tr1.SetTailColor(Color.green);

**else**

tr1.SetTailColor(Color.blue);

Draw\_square(tr1, 100);

tr1.TurnRight(10);

}

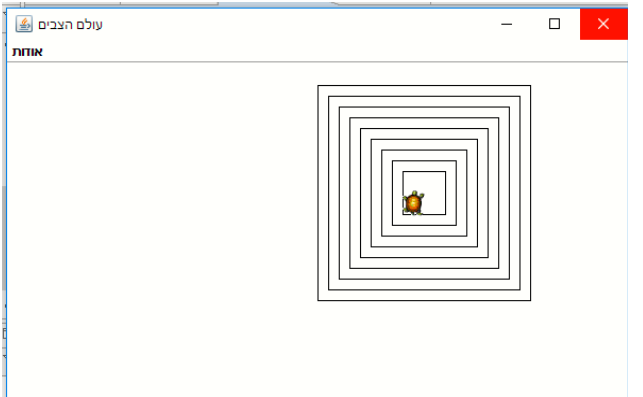
}

}



משימה 5: ריבועים מתכנסים

**כתבו** אלגוריתם המצייר באמצעות הצב את הציור הבא:



זהו ריבוע שאורך צלעו X צעדי צב. בתוכו ריבוע נוסף שאורך צלעו X-20 צעדי צב

ובתוכו ריבוע נוסף שאורך צלעו X-20 צעדי צב מהריבוע שקדם לו וכן הלאה.

הצב יפסיק לצייר כאשר אורך הצלע שלו קטנה מ-30 צעדי צב.

לפתרון הבעיות שראינו עד כה נעזרנו בלולאת for שחזרה על עצמה מספר פעמים מוגדר מראש על פי המונה שהגדרנו. לא חובה לקבוע מראש בתוכנית את מספר הפעמים שהלולאה תתבצע, ואפשר לקלוט ערך זה מהמשתמש.

בפתרון הבעיה הנוכחית קשה לדעת מראש מה יהיה מספר הסיבובים בלולאה, ולכן השימוש במונה אינו מתאים. נדרש כאן שימוש בתנאי, ועלינו לבדוק בכל שלב האם עדיין יש צורך לחזור ולבצע את הלולאה בהתאם לתנאי. במקרים שבהם ביצוע הלולאה תלוי בתנאי מתאים יותר מבנה לולאה אחר.

לשם כך נכיר הוראת חזרהנוספת,המכונה **לולאת while**. בלולאת while אין משתנה בקרה שגדל בכל שלב, אלא מוגדר תנאי להמשך הביצוע החוזר שלה.

המבנה הכללי של לולאת while הוא:

while (ביטוי בוליאני) {

גוף הלולאה

}

כעת נחזור ליישם את המבנה במסגרת משימה 5:

אלגוריתם לפתרון הבעיה ריבוע בתוך ריבוע

**ריבוע בתוך ריבוע**

נגדיר צב שמצייר את הציור. נקלוט מספר שלם וחיובי למשתנה x.

כל עוד מתקיים x>30, נחזור ונבצע את ההוראות הבאות:

הצב יצייר ריבוע שאורך צלעו x. לאחר מכן נפחית מערכו של המשתנה x את הערך 20.

נחשב את מיקומו החדש של הצב ונעביר אותו מבלי להשאיר סימן על המסך למקום חדש.

מימוש בסביבת העבודה על ידי לולאת while:

**import** java.util.Scanner;

**import** unit4.turtleLib.Turtle;

**public** **class** Ex5

{

**public** **static** **void** drow\_rectangle(Turtle dr\_tr1 , **int** x )

{

**for**(**int** k =1; k<=4 ; k++)

{

dr\_tr1.moveForward(x);

dr\_tr1.turnRight(90);

}

}

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

{

Turtle tr1 = **new** Turtle();

Scanner sc = **new** Scanner(System.in) ;

tr1.tailDown();

tr1.setDelay(100);

System.out.println("Enter size of sqaure

between 90....200 ");

**int** length = sc.nextInt() ;

**while** (length>=30)

{

drow\_rectangle(tr1, length);

tr1.tailUp();

tr1.moveForward(10);

tr1.turnRight(90);

tr1.moveForward(10);

tr1.turnLeft(90);

tr1.tailDown();

length = length -20 ;

}

}

}

C#

using System;

using Unit4.TurtleLib;

**class** Ex5

{

**static** **void** Draw\_rectangle(Turtle dr\_tr1 , **int** x )

{

**for**(**int** k =1; k<=4 ; k++)

{

dr\_tr1.MoveForward(x);

dr\_tr1.TurnRight(90);

}

}

**static** **void** Main(string[] args)

{

Turtle tr1 = **new** Turtle();

tr1.TailDown();

tr1.SetDelay(100);

Console.WriteLine("Enter size of sqaure

between 90....200 ");

**int** length = int.Parse(Console.ReadLine()); **while** (length>=30)

{

Draw\_rectangle(tr1, length);

tr1.TailUp();

tr1.MoveForward(10);

tr1.TurnRight(90);

tr1.MoveForward(10);

tr1.TurnLeft(90);

tr1.TailDown();

length = length - 20 ;

}

}

}

**רגע חושבים!**

לפניכם קטע תוכנית. מה הפלט שלה? וכמה פעמים מתבצעת הלולאה?

int num = 10;

int sum = 0;

for (int j = 1; j <= 10; j++)

sum = sum + counter;

System.out.println("end loop sum = " + sum);

}

C#

int num = 10;

int sum = 0;

for (int j = 1; j <= 10; j++)

sum = sum + counter;

Console.WriteLine("end loop sum = " + sum);

נראה כעת כי ניתן לכתוב קטע זה בדרכים רבות. לאחר שנציג אותן, נגיע למסקנה מפתיעה:

גוף הלולאה נתון בתוך { }

int num = 10;

int sum = 0;

for (int counter = 1; counter <= 10; counter++) {

sum = sum + counter;

}

System.out.println("end loop sum = " + sum);

C#

int num = 10;

int sum = 0;

for (int counter = 1; counter <= 10; counter++) {

sum = sum + counter;

}

Console.WriteLine("end loop sum = " + sum);

אתחול מונה הלולאה נעשה מחוץ למסגרת שלה

int num = 10;

int sum = 0;

int counter = 1;

for (; counter <= 10; counter++) {

sum = sum + counter;

}

System.out.println("end loop sum = " + sum);

C#

int num = 10;

int sum = 0;

int counter = 1;

for (; counter <= 10; counter++) {

sum = sum + counter;

}

Console.WriteLine("end loop sum = " + sum);

קידום מונה הלולאה נרשם בתוך גוף הלולאה

int num = 10;

int sum = 0;

int counter = 1;

for (; counter <= 10;) {

sum = sum + counter;

counter++

}

System.out.println("end loop sum = " + sum);

C#

int num = 10;

int sum = 0;

int counter = 1;

for (; counter <= 10;) {

sum = sum + counter;

counter++

}

Console.WriteLine("end loop sum = " + sum);

איזו לולאה מזכיר קטע זה?

סיכום: הכרנו שתי הוראות חזרה, שנקראות לולאות: לולאת while ולולאת for.

גילינו כי שני סוגי הלולאות מסייעים בפתרון מצבים שבהם עלינו לחזור ולבצע סדרה של הוראות.

בעבודה עם לולאות אנו נדרשים לעיתים קרובות להגדיר משתנה **מונה** או **צובר**.

**מונה** – תפקידו למנות את מספר הפעמים שהתרחש אירוע מסוים, למשל מספר הפעמים שנקרא נתון קלט. כיוון שהמונה מבצע ספירה, משתנה זה יהיה מטיפוס שלם.

**צובר** – תפקידו לצבור ערכים, וניתן להשתמש בו למשל לחישוב סכום.

כך ראינו בדוגמה האחרונה, שבה חושב סכום הערכים מ-1 ועד ערכו של המשתנה num.

אתחול של צובר או מונה מתבצע כאשר נזין לתוכם ערך מתאים לתחילת התהליך.

מונה מאותחל בדרך כלל בערך הראשוני 0. ערכו הראשוני של צובר תלוי במהות הצבירה המתבצעת בו – למשל, צובר ששומר סכום מצטבר יאותחל ב-0.

צובר נפוץ נוסף הוא **צובר של מחרוזות**.



משימה 6: צובר מחרוזת

**כתבו** אלגוריתם המקבל מחרוזת st ומספר שלם num. האלגוריתם מחזירה מחרוזת חדשה, שבה המחרוזת st משורשרת num פעמים.

לדוגמה: הפעולה מקבלת את המחרוזת "agr" ואת המספר 4, ומחזירה את המחרוזת agragragragr.

**ממשו** את האלגוריתם בפעולה הבאה:

public static String concatStr(String st, int num)

תנו דוגמה לצובר שיאותחל בערך שונה. מדוע?

**לולאה אינסופית:**

נתון קטע תוכנית. הקלט הוא מספר שלם חיובי.

int num = קלט למשתנה

while (num != 0) {

num = num % 3;

System.out.println(num);

}

C#

int num = קלט למשתנה

while (num != 0) {

num = num % 3;

Console.WriteLine(num);

}

1. תנו דוגמה לקלט שעבורו תתבצע הלולאה מספר סופי של פעמים.
2. מה מאפיין את הקלטים שעבורם תתבצע הלולאה מספר סופי של פעמים?
3. כמה פעמים תתבצע הלולאה עבור קלטים אלה?
4. תנו דוגמה לקלט שעבורה הלולאה לא תסתיים.
5. מה מאפיין את הקלטים שעבורם הלולאה תחזור ותתבצע שוב ושוב בלי להסתיים?

הלולאה בקטע התוכנית תתבצע מספר סופי של פעמים עבור חלק מן הקלטים, ומספר אינסופי של פעמים עבור שאר הקלטים. לכן הלולאה שבקטע התוכנית היא לולאה אינסופית.

בפיתוח אלגוריתם קורה לעתים שנכתבת לולאה אינסופית. לולאה כזו עלולה להתבצע אינסוף פעמים רק עבור חלק מן הקלטים. בחומר הלימוד שלנו תוכניות הכוללות לולאה אינסופית נחשבות לשגויות.

**רגע חושבים!**

האם לולאה אינסופית היא בהכרח תקלה? האם קיימים מקרים שבהם אנו רוצים שלולאה תהיה אינסופית?

**צפו** בסרטון הבא – [Echo/Google home infinite loop](https://www.youtube.com/watch?v=ZfCfTYZJWtI)

בפרק זה הכרנו שתי לולאות שבעזרתן ניתן לפתור בעיות אלגוריתמיות.

נסכם חלק זה של הפרק בשתי משימות שבהן תוכלו להשתמש בכל אחת מהלולאות לפי בחירתכם.

### משימה ראשונה: מרוץ הצבים – תחרות ריצה בין שני צבים

ניגש לפירוק המשימה לתתי-משימות:

* הכנת המרוץ: יש לצייר את קו הזינוק ואת קו סיום. המרחק בין הקווים יהיה 400 צעדים.

נגדיר צב שתפקידו לצייר. בסיום המשימה הוא ייעלם מהמסך.

| |

| |

| | זינוק

| |

| |

* הגדרת שני צבים המתחרים במרוץ, העברתם לנקודת הזינוק והעמדתם על קו הזינוק במרחק זה מזה כשפניהם לכיוון מסלול הריצה.
* המרוץ – כל עוד שני הצבים לא חצו את קו הסיום, יש לבצע את ההוראות הבאות:

הגרלת צעדי צב בין 40-10 עבור שני הצבים

ביצוע הריצה לשני הצבים

* הכרזה על הזוכה (האם ייתכן מצב של תיקו?).

### משימה שנייה: מספרים ראשוניים

**למה הם מעניינים אותנו?**

מספר ראשוני הוא מספר הוא מספר טבעי גדול מ-1, שלא ניתן להציגו כמכפלה של שני מספרים טבעיים קטנים ממנו, כלומר הוא מתחלק רק ב-1 ובעצמו. המספר 1 בעצמו אינו נחשב ראשוני. מספרים ראשוניים חשובים מאוד עבור הצפנות למשל.

משחר ההיסטוריה מנסים בני האדם להעביר הודעות סודיות זה לזה, כך שהמסרים יהיו מובנים רק לשני הצדדים המעורבים בשיחה ולא לאף אחד אחר. הסתרת מסרים מפני קוראים פוטנציאליים נקראת **הצפנה**. היום הנושא חשוב יותר מאי פעם בגלל סוגיות של אבטחת מידע אישי ברשתות האינטרנט. למשל, אנו רוצים לנהל חשבון בנק דרך האינטרנט, ואיננו רוצים שכל אחד יוכל לראות את מצבנו הכספי ואת הפעולות שאנו עושים בחשבוננו; אנו רוצים לבצע תשלומים באינטרנט; וכן הלאה. פעולות אלה דורשות תקשורת מוצפנת בין שני צדדים או יותר.

במהלך לימודי מדעי המחשב תוכלו להעשיר את הידע שלכם בהיכרות עם מושגים חשובים ושימושיים מעולם ההצפנה. להבנה טובה יותר של הצפנה צפו בסרטון הבא: [Asymmetric encryption – Simply explained](https://www.youtube.com/watch?v=AQDCe585Lnc). להעמקה נוספת אתם מוזמנים לעיין בסקירות העוסקות בהסבר כללי כיצד פועלת הצפנה, ובאופן ספציפי בהצפנה א-סימטרית.

<http://meyda.education.gov.il/files/Tochniyot_Limudim/Math/Hatab/Hatzpana1.pdf>

<http://meyda.education.gov.il/files/Tochniyot_Limudim/Math/Hatab/Hatzpana2.pdf>

**פתחו** אלגוריתם שמקבל מספר ומחזיר true אם המספר ראשוני, אחרת מחזיר false.

**ממשו** את האלגוריתם בסביבת העבודה.

כתבו תוכנית המדפיסה את כל המספרים הראשוניים מ-1 ועד 100,000.

**לפני שממשיכים הלאה – תרגיל נוסף בלולאת for:**

נוכל להגדיר משתנה בלולאה כך:

for (int j = 1; j <= 5; j++) {

System.out.print("\*");

}

System.out.println("end loop j = " + j); 🡨 error

C#

for (int j = 1; j <= 5; j++) {

Console.Write("\*");

}

Console.WriteLine("end loop j = " + j); 🡨 error

האם ניתן להדפיס את ערכו של המשתנה j בסיום הלולאה?

הבעיה היא שהמשתנה מוכר אך ורק במסגרת הלולאה. ברגע שיוצאים ממנה הוא אינו מוכר, ולכן לא ניתן לפנות אליו.



משימה 8: לולאה מקוננת

**כתבו** פעולה בשם example1 שמקבלת מספר שלם בין 4 ל-9, ומחזירה פלט מדורג של טוח המספרים מ-1 ועד מספר הקלט. עבור המספר 5, למשל, הפעולה מדפיסה את הפלט הבא:

**example1**

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

**ניגש לניתוח הבעיה:**

המשימה שלנו מורכבת – אנו צריכים להדפיס n שורות, ובכל שורה צריך להדפיס מספר שונה של ספרות.

האם ניתן לבצע את המשימה על ידי שימוש בלולאה אחת?

נגדיר לולאה שמבצעת הדפסה של חמש שורות. בשלב זה לא נטפל בהדפסת השורה עצמה.

public static void example1(int n) {

for (int row = 1; row <= n; row++) {

הדפסת שורה

}

}

C#

public static void Example1(int n) {

for (int row = 1; row <= n; row++) {

הדפסת שורה

}

}

עכשיו נטפל בבעיה של הדפסת השורה. לא נוכל להשתמש במונה השורות כיוון שהוא רק מצביע על מספר השורה, ולכן חייבים להגדיר מונה נוסף. בכל שורה תמיד מתחילים למנות מ-1.

הבעיה היא שבכל שורה מספר הפעמים שאנו מדפיסים שונה. כרגע נרשום **????**

for (int k = 1; k <= ? ? ? ? ? ; k++)

System.out.print( ? ? ? )

C#

for (int k = 1; k <= ? ? ? ? ? ; k++)

Console.Write( ? ? ? )

איזה מספר נדפיס? אם נדפיס את ערכו של המשתנה k, לא נקבל את הפלט הרצוי. מה לדעתכם יקרה אם נדפיס את ערכו של המשתנה row?

בשורה הראשונה יודפס 1 (מספר פעמים ...)

בשורה השנייה יודפס 2 (אותו מספר פעמים...)

וכן הלאה

נראה שהצלחנו לפתור חלק מהבעיה, ולכן כעת נשלב את שתי הלולאות יחדיו:

public static void example1(int n) {

for (int row = 1; row <= n; row++) {

for (int k = 1; k <= ? ? ? ? ? ; k++)

System.out.print(row);

}

}

C#

public static void Example1(int n) {

for (int row = 1; row <= n; row++) {

for (int k = 1; k <= ? ? ? ? ? ; k++)

Console.Write(row);

}

}

אך עדיין לא רשמנו את התנאי בלולאה הפנימית. האם k<=n יפתור לנו את הבעיה? **נמקו** תשובתכם.

**רמז**: כמה מספרים יודפסו בשורה הראשונה?

**התבוננו** בטבלה הבאה ונסו להבין ממנה מתי הלולאה הפנימית צריכה להסתיים, באיזה משתנה אנו תלויים .....k<= row

public static void example1(int n) {

for (int row = 1; row <= n; row++) {

for (int k = 1; k <= row; k++)

System.out.print(row);

}

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| פלט | K<= | K | Row<=n | row |
| 1 |  | 1 | 1<=5 true | 1 |
| 2 |  | 1 | 2<=5 | 2 |
| 2 |  | 2 |  |  |
| 3 |  | 1 | 3<=5 | 3 |
| 3 |  | 2 |  |  |
| 3 |  | 3 |  |  |

C#

public static void Example1(int n) {

for (int row = 1; row <= n; row++) {

for (int k = 1; k <= row; k++)

Console.Write(row);

}

}

 שימו לב

הפלט המתקבל יהיה כולו בשורה אחת, כי השתמשנו בפקודת הדפסה בלי לרדת שורה. אם נשתמש בפקודת הדפסה לירידת שורה, נקבל כל מספר בשורה נפרדת.  
על מנת לקבל את הפלט כפי שהוא מוגדר במשימה, עלינו לרדת שורה רק לאחר שסיימנו להדפיסה.

public static void example1(int n) {

for (int row = 1; row <= n; row++) {

for (int k = 1; k <= row; k++)

System.out.print(row);

System.out.println();

}

}

C#

public static void Example1(int n) {

for (int row = 1; row <= n; row++) {

for (int k = 1; k <= row; k++)

Console.Write(row);

Console.WriteLine();

}

}

 שימו לב

הלולאה הפנימית מבצעת הוראה אחת בלבד – הוראת הדפסה, לכן לא חייבים להוסיף סוגריים מסולסלים.

לעומת זאת הלולאה החיצונית אנו זקוקים בהכרח לסוגרים מסולסלים, כי היא מכילה שתי הוראות לביצוע – לולאה והוראת הדפסה.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| פלט | K<= row | | K | Row<=n | | | row |
| 1 | 1<=1 true | | 1 | 1<=5 true | | | 1 |
|  | 2<=1 false | | 2 |  | | |  |
| 2 | 1<=2 true | | 1 | 2<=5 | | | 2 |
| 2 | 2<=2 true | | 2 |  | | |  |
|  | 3<=2 false | | 3 |  | | |  |
| 3 |  | | 1 | 3<=5 | | | 3 |
| 3 |  | | 2 |  | | |  |
| 3 |  | | 3 |  | | |  |
|  |  | |  |  | | |  |
|  | |  |  | |  |  | | |  |

לולאה המכילה לולאה נוספת בתוכה קרויה לולאה מקוננת nested)). לתרגול נוסף של הלולאה המקוננת, גשו לתרגילים בסיום הפרק העוסקים בנושא זה.

בהמשך מופיעים שבעה פלטים נוספים. עבור כל אחד מהם יש לכתוב פעולה מתאימה.

**example 2**

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

**example 3**

5 5 5 5 5

4 4 4 4

3 3 3

2 2

1

**example 4**

5 4 3 2 1

5 4 3 2

5 4 3

5 4

5

**example 5**

5

5 4

5 4 3

5 4 3 2

5 4 3 2 1

**example 6**

5

4 5

3 4 5

2 3 4 5

1 2 3 4 5

**example 7**

1 2 3 4 5

2 3 4 5

3 4 5

4 5

5

**example 8**

5 4 3 2 1

4 3 2 1

3 2 1

2 1

1

כתבו שמונה פעולות שכל אחת מהן מקבלת כפרמטר מספר שלם ומדפיסה את הדוגמה הנדרשת

public void example1(int n)

public void example2(int n)

C#

public void Example1(int n)

public void Example2(int n)

##### סיכום פרק תנאים ומחלקות עזר

* **לולאת for** המבנה הכללי של ההוראה:

for(אתחול משתנה הבקרה ; התנאי להמשך הביצוע ; שינוי משתנה הבקרה)

{

הוראות לביצוע

}

* **אתחול משתנה הבקרה:** הוראת השמה שקובעת ערך התחלתי למשתנה הבקרה.
* **התנאי להמשך הביצוע:** ביטוי בוליאני שמשמש תנאי לביצוע חוזר. התנאי נבדק לראשונה אחרי אתחול משתנה הבקרה, ונבדק שוב בכל פעם שמסתיים ביצוע של קבוצת הוראות. כל עוד התנאי מקבל ערך true, הביצוע החוזר ממשיך. כאשר ערכו של התנאי הוא false, הביצוע החוזר מסתיים.
* **שינוי משתנה הבקרה:** שינוי שחל במשתנה הבקרה בכל פעם שמסתיים שלב ביצוע נוסף. גוף הלולאה תחום בסוגריים מסולסלים. אם גוף הלולאה מכיל משפט בודד אפשר להשמיט את הסוגריים.
* **לולאת while** המבנה הכללי של ההוראה

while ( ביטוי בוליאני המווה תנאי להמשך הביצוע )

{

גוף הלולאה : משפט או משפטים לביצוע

}



משימה מסכמת: לא עוצרים לפני 100

במשחק שני שחקנים או יותר, ולרשות כל השחקנים זוג קוביות.

כל שחקן בתורו מטיל את הקוביות, כל תור במשחק מקבל מספר מ-1 ועד 6.

השחקן צובר נקודות בהתאם למספר התור ולמספרים העולים בקוביות.

המשחק מסתיים כאשר מספר הנקודות שצבר אחד מהשחקנים הוא 100 לפחות.

דוגמאות:

תור מספר **2** : אם התוצאה בקוביות **2:2** השחקן מקבל 21 נקודות

אם יצא באחת מהקוביות **2** השחקן מקבל **2** נקודות

אם בקוביות לא יצא מספר 2, השחקן מקבל 0 נקודות.

תור מספר **5**: אם התוצאה בקוביות **5:5** השחקן מקבל 21 נקודות

אם יצא באחת מהקוביות **5**, השחקן מקבל **5** נקודות

אם בקוביות לא יצא מספר 5 השחקן מקבל 0 נקודות.

להלן הדמיה של מהלכי המשחק עבור שני שחקנים:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר התור | הטלת הקוביות | הנקודות שקיבל השחקן בתור | ניקוד מצטבר שחקן1 | ניקוד מצטבר שחקן2 |
|  |  |  | 0 | 0 |
| 1 | 5:3 | 0 | 0 |  |
| 2 | 4:4 | 0 |  | 0 |
| 3 | 6:3 | 3 | 3 |  |
| 4 | 4:1 | 4 |  | 4 |
| 5 | 4:4 | 0 | 3 |  |
| 6 | 6:1 | 6 |  | 10 |
| 1 | 1:1 | 21 | 24 |  |
| 2 | 2:5 | 2 |  | 12 |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |

כתבו אלגוריתם למשחק **לא עוצרים לפני 100**.

על מנת שנוכל לכתוב אלגוריתם למשחק, עלינו לפרק אותו למשימות פשוטות יותר:

א . הגדרת המשתנים ואתחולם.

ב . תיאור מהלך בודד של המשחק.

ג . חזרה על המהלך הבודד של המשחק כל עוד איש מהשחקנים לא צבר 100 נקודות או יותר.

ד. הכרזה על השחקן המנצח.

**א . הגדרת המשתנים**:

משתנה למניית התורים – counter

כל שחקן צריך משתנה הצובר בתוכו את הנקודות בכל מהלך: points1 , points2

עצם שיאפשר לנו לבצע הדמיה להגרלת קוביות.

שני משתני עזר לשמירת ערכי הגרלת הקוביות: die1 , die2

נחשוב רגע על מניית התורים: המנייה מתחילה ב-1 וממשיכה עד 8. אבל במשחק נאמר שאחרי תור מספר 6 מספר התור חוזר להיות 1.

משתנה נוסף יכול לעזור לנו. נקרא לו rounds. באתחול הוא יקבל את הערך 1, ובכל תור חדש ערכו יעלה ב 1. כאשר ערכו יהיה 7, הוא יקבל את הערך 1 שוב

(האם חייבים להשתמש במשתנה נוסף?...)

**ב . תיאור של מהלך בודד**:

נניח שמספר המהלך הוא round.

נבדוק אם ערכו הוא 7, אם כן נשים בו את הערך 1.

אם בכל אחת מהקוביות יצא הערך round' נוסיף לשחקן 21 נקודות בנוסף לנקודות שצבר עד כה.

אם רק באחת מהקוביות יצא הערך round' נוסיף לשחקן round נקודות בנוסף לנקודות שצבר עד כה.

**ג . חזרה על מהלך בודד עד אשר אחד השחקנים צבר 100 נקודות או יותר**

כל שחקן בתורו מטיל את הקוביות. כיצד נדאג לכך שבכל סיבוב של המשחק נפנה לשחקן אחר? האם צריך משתנה נוסף?

**ד. הכרזה על השחקן המנצח**

בסיום המשחק צריך לבדוק מי השחקן שמספר הנקודות שצבר שווה ל-100 או גדול מ-100, והוא יוכרז כשחקן המנצח.

### תרגילים – לולאת for

עבור כל אחת מהבעיות הבאות עליכם לפתח אלגוריתם – כלומר לכתוב אלגוריתם וליישם אותו בסביבת העבודה. בכל בעיה ניתן להניח כי הקלט המתקבל הוא תקין ועונה לדרישות המתוארות. על האלגוריתם להדפיס את הפלט הנדרש.

1 . **סכום חכם:**

**קלט:** מספר שלם וחיובי .

**פלט:** כל המספרים החל מ- ועד המתחלקים ב-7 ללא שארית.

2 . **סדרה ללא שארית:**

**קלט:** מספר שלם וחיובי .

**פלט:** כל המספרים החל מ- ועד 1 בסדר יורד, המתחלקים ב-4 ללא שארית או שמתחלקים ב-3 ללא שארית.

3 . **חיבור וכפל:**

**קלט:** שני מספרים שלמים וחיוביים .

**פלט:** תוצאת הכפל .

**הדרכת חובה:** יש לחשב את תוצאת הכפל בעזרת פעולת חיבור בלבד.

4 . **כפל וחזקה:**

**קלט:** שני מספרים שלמים וחיוביים

**פלט:** החזקה .

**הדרכת חובה:** יש לחשב את תוצאת העלאת החזקה בעזרת פעולת כפל בלבד.

5 . **עוצרים!**

**קלט:** מספר שלם אי-שלילי .

**פלט:** עצרת המספר: .

**דוגמה:** עבור הקלט 4 הפלט יהיה 24, לפי החישוב הבא:

**תזכורת:** ההגדרה המתמטית עבור אפס עצרת היא :

6. **סדרת פיבונצ'י:**

סדרת פיבונצ'י מוגדרת באופן שונה מעט ממה שפגשתם בפרק. נגדיר אותה כך:

האיבר הראשון הוא 0

האיבר השני הוא 1

האיבר השלישי ואילך הוא סכום שני האיברים הקודמים לו.

כך מתקבלת הסדרה:

**קלט:** מקום של איבר בסדרה

**פלט:** האיבר בסדרה במקום המתאים.

7 . **תולעת חרוצה:**

תולעת מטפסת על קיר במשך כמה ימים. ביום הראשון היא מתקדמת 1 ס"מ.

ביום השני היא מתקדמת 3 ס"מ, ביום השלישי 9 ס"מ וכן הלאה.

בכל יום מוכפל המרחק שהיא עוברת פי 3 מהמרחק שעברה ביום הקודם.

הניחו שהקיר שעליו מטפסת התולעת הוא אינסופי.

**קלט:** מספר שלם גדול או שווה לאחד, הוא מספר הימים.

**פלט:** המרחק המצטבר שעברה התולעת במשך הימים.

**דוגמה:** עבור ארבעה ימים, המרחק הוא

8 . **בדיקת סדרה עולה:**

סדרת מספרים נקראת סדרה עולה אם לכל זוג איברים עוקבים בסדרה, האיבר הראשון בזוג קטן מהאיבר הבא אחריו.

**קלט:** סדרה לא ריקה של מספרים שלמים.

**פלט:** “Yes” אם סדרת המספרים עולה ו-“No” אחרת.

**הדרכת חובה:** בשלב ראשון יש לקלוט את מספר האיברים בסדרה, ולאחר מכן לקלוט את איברי הסדרה עצמם.

 שימו לב

סדרה ובה מספר יחיד מוגדרת כסדרה עולה.

### תרגילים – לולאת while

עבור כל אחת מהבעיות הבאות עליכם לפתח אלגוריתם – כלומר לכתוב וליישם אותו בסביבת העבודה. בכל בעיה ניתן להניח כי הקלט המתקבל הוא תקין ועונה לדרישות המתוארות בקלט. על האלגוריתם להדפיס את הפלט הנדרש.

1 . **גובה הכיתה:**

**קלט:** גבהים של תלמידים בכיתה.

**פלט:** ממוצע הגבהים של תלמידים.

**הדרכה:** יש לסיים את קליטת הגבהים באמצעות הזנת מספר שלילי או 0.

2 . **זוגות-זוגות:**

**קלט:** מספר חיובי זוגי .

**פלט:** כל המספרים החיוביים הזוגיים הקטנים או שווים ל-.

**דוגמה:** עבור הקלט 8 הפלט הוא .

3 . **לפרק את הסכום:**

**קלט:** מספר שלם חיובי

**פלט:** רשימה עולה גדולה ביותר של מספרים שלמים המתחילה ב-1, אשר סכומם המצטבר אינו עולה על המספר

**דוגמה:** עבור הקלט 20 הפלט יהיה 1 2 3 4 5, כיוון שהסכום 1+2+3+4+5 = 15 . אם נוסיף 6 נגיע ל-21, ולכן 6 לא יודפס.

4 . **חיסור מקסימלי:**

**קלט:** סדרה של מספרים שלמים, חיוביים ושליליים.

**פלט:** ההפרש הגדול ביותר בין שני מספרים בסדרת הקלט.

**הדרכה:** יש לסיים את קליטת המספרים במספר 0.

**הדרכת חובה:** על הסדרה להיות בת שני מספרים לפחות, לא כולל המספר 0.

5 . **אחדות ועשרות:**

**קלט:** סדרה של מספרים שלמים, חיוביים ושליליים.

**פלט:** מונה המציין כמה מספרים בסדרה מקיימים את התנאי הבא:

ספרת האחדות זהה לספרת העשרות.

**הדרכה:** יש לסיים את קליטת המספרים במספר 0.

6 . **עד 2 בחזקה:**

**קלט:** מספר חיובי שלם .

**פלט:** תוצאת חישוב החזקה הקטנה ביותר של 2 אשר גדולה מ-.

**דוגמאות:**

* עבור הקלט 7 הפלט יהיה 8, כיוון ש-
* עבור הקלט 8 הפלט יהיה 16, כיוון ש-

### לולאתwhile – תרגיל מעקב

1 . **סופי או אינסופי:**

ציינו עבור כל אחת מהלולאות הבאות אם היא סופית או אינסופית.

אם הלולאה סופית ציינו כמה פעמים היא מתבצעת.

אם הלולאה סופית ציינו מהו ערכו של משתנה הלולאה לאחר סיומה.

חובה להציג טבלת מעקב לפי הדוגמה המצורפת לתרגיל 1.1

עבור כל לולאה, הניחו כי מוגדרים שלושה משתנים מטיפוס שלם בתחילת הקוד הנתון

int i, j, k;

**1.1**

i = 0;

while (i < 30)

i = i + 5;

System.out.println("\*\*\*");

C#

i = 0;

while (i < 30)

i = i + 5;

Console.WriteLine("\*\*\*");

**1.2**

j = 0;

while (j < 50)

j = j - 10;

System.out.println("\*\*\*");

C#

j = 0;

while (j < 50)

j = j - 10;

Console.WriteLine("\*\*\*");

**1.3**

k = 0;

j = 1;

while (j <= 5) {

k = k + 11;

j++;

}

System.out.println("\*\*\*");

C#

k = 0;

j = 1;

while (j <= 5) {

k = k + 11;

j++;

}

Console.WriteLine("\*\*\*");

2. **הכול תלוי בקלט:**

נתונים שני קטעי תוכניות. לכל קטע תוכנית **תנו דוגמה** לקלט שעבורו הלולאה סופית, ושתי דוגמאות קלט שעבורן הלולאה היא אינסופית.

**חשבו** על דוגמאות המייצגות קבוצות קלט שונות זו מזו.

**2.1**

int num;

Scanner reader = new Scanner(System.in);

num = reader.nextInt();

while (num != 10) {

System.out.print(num + " ");

num = num + 2;

}

C#

int num;

num = int.Parse(Console.ReadLine());

while (num != 10) {

Console.Write(num + " ");

num = num + 2;

}

**2.2**

int num;

num = int.Parse(Console.ReadLine());

while (num != 0) {

Console.Write(num + " ");

num = num - 2;

}

C#

int num;

Scanner reader = new Scanner(System.in);

num = reader.nextInt();

while (num != 0) {

System.out.print(num + " ");

num = num - 2;

}

### תרגיל ארוך – פירוק מספרים

בכל ארבעת התרגילים הבאים יש להניח כי הקלט הוא מספר שלם וחיובי גדול מאוד.

1 . **מספר הספרות**

**קלט:** מספר שלם וחיובי .

**פלט:** מספר הספרות ב-.

**דוגמה:** עבור הקלט 120451 הפלט יהיה 6.

2 . **סכום הספרות**

**קלט:** מספר שלם וחיובי.

**פלט:** סכום הספרות במספר.

**דוגמה:** עבור הקלט 123456 הפלט יהיה 21.

3 . **הדפסת היפוך המספר**

**קלט:** מספר שלם וחיובי.

**פלט:** המספר ההפוך למספר שנקלט.

**דוגמה:** עבור הקלט 123456 הפלט יהיה 654321.

**הדרכה:** מותר לשנות את המספר שנקלט.

4 . **בניית מספר הפוך**

**קלט:** מספר שלם וחיובי

**פלט**: המספר והמספר ההפוך ל-

**דוגמה:** עבור הקלט 1234 הפלט יהיה המספר המקורי 1234 והמספר ההפוך יהיה 4321.

**הדרכת חובה**: יש לשמור את המספר המקורי במשתנה עזר .



תרגילי אתגר

**1 . מקום שני**

**קלט:** גודל רשימת המספרים הנקלטים ורשימה של מספרים שלמים.

**פלט:** המספר **השני בגודלו** ברשימת המספרים.

**הדרכת חובה:** יש לקלוט את גודל הרשימה לפני ביצוע הקלט לרשימה עצמה. המספרים בקלט שונים זה מזה.

**דוגמה:** עבור הקלט משמאל לימין 7 17 -4 190 -30 60 25 9

המספר הראשון בקלט הוא 7, המציין את מספר האיברים ברשימה.

הפלט בדוגמה יהיה 60.

**2 . אחד ברצף**

**קלט:** רשימה של מספרים שלמים 0 ו- 1 בלבד.

**פלט:** אורכו של הרצף הגדול ביותר של המספר 1, והמקום שבו מתחיל הרצף

**הדרכה:** יש לסיים את קליטת המספרים במספר 9.

**דוגמאות:**

* עבור רשימת הקלט משמאל לימין 0100111111001111019

הפלט יהיה 6, 5 כיוון שאורכו של הרצף הגדול ביותר הוא 6 ותחילתו במקום 5.

* עבור רשימת הקלט משמאל לימין 000009

הפלט יהיה 0, 0 כיוון שאורכו של הרצף הגדול ביותר הוא 0 (אין אחדות בקלט). נסמן זאת במיקום התחלתי שלא קיים, והוא 0.

**3 . מחרוזת PN של מספרים**

**קלט:** זוגות של מספרים שלמים, כך שכל זוג מורכב ממספרים חיוביים או שליליים.

**פלט:** הדפסת המחרוזת הנוצרת בהתאם להדרכה.

**הדרכה**: יש לסיים את קליטת המספרים במספר 0.

כל זוג מספרים בקלט נבדק.

אם שני המספרים בעלי אותו סימן (שניהם חיוביים או שניהם שליליים), מוסיפים למחרוזת מימין את התו P. אחרת מוסיפים למחרוזת מימין את התו N.

**4 . פירוק לגורמים**

**קלט:** מספר שלם וחיובי הגדול מ-1 .

**פלט:** הגורמים הראשוניים של לפי סדר גודלם, מקטן לגדול, כולל חזרות.

**דוגמאות:**

* עבור הקלט 7 הפלט יהיה 7
* עבור הקלט 9 הפלט יהיה 3 3
* עבור הקלט 60 הפלט יהיה 2 2 3 5

#### פרק 5: מבני נתונים סדרתיים

הקדמה

בפרקים הקודמים הכרנו משתנים מטיפוסים שונים ופיתחנו אלגוריתמים מגוונים. אולי הבחנתם בכך שבמרבית התכניות שבנינו עד כה לא נזקקנו למשתנים רבים ולשמירת המידע באופן מסודר. בפרק זה נפתח אלגוריתמים אשר יעשו שימוש במספר רב של משתנים ויטפלו במידע שיש בו סדר מסוים. כאן נפגוש לראשונה את תחום **מבני הנתונים** שבו נתעמק בשער השני של הספר.

**רגע חושבים!**

אילו אפליקציות אתם מכירים שצריכות לשמור נתונים רבים, כמו מסרונים, נתונים גאוגרפיים, תמונות, סרטונים וכדומה? האם יש מבנה או סדר כלשהו שאתם יכולים לחשוב עליו בניהול הנתונים הללו?

בפרק זה נעמיק יותר בשאלה זו.

התחילו במשימה הבאה:



### משימה 1

צפו בסרטון [Behind the scenes of an Amazon warehouse](https://www.youtube.com/watch?v=Y-lBvI6u_hw) ב-youtube.

**השיבו על** השאלות הבאות:

1. כדי שתוכנת הניהול תשלוט בכל רובוט, כמה משתנים יש להגדיר לדעתכם?
2. איך התוכנה זוכרת היכן נמצא כל פריט במחסן? האם מספר הפריטים משפיע על מספר המשתנים שיש בתוכנה?
3. איך התוכנה מוצאת ללקוחות את הפריט שהם מחפשים?

קשה ומסורבל לנהל אוסף גדול של נתונים באמצעות כלי התכנות שאיתם עבדנו עד כה.

אחת השיטות לשמירת נתונים שמאפשרת גישה מהירה אליהם היא מבנה נתונים שנקרא **מערך** (array).

מערך פשוט יכול לסייע אפילו בניהול המחסן של אמזון. צפו בסרטון [Lotte vertical parking system](https://www.youtube.com/watch?v=RB1dQMbfMGA) ב-youtube.

**השיבו** על השאלות הבאות:

1. איך מתקבלת ההחלטה היכן למקם את המכונית?
2. כאשר מכונית חונה בחניה, איזה נתון נשמר במחשב?
3. איך אפשר לשמור את הנתונים לכל המכוניות בחניון?

כפי שראיתם בסרטון, המערכת יודעת למקם את הרכב במקום פנוי, לשמור את מיקומו, למצוא אותו ולהוציא אותו החוצה כאשר בעל הרכב מבקש לחזור לרכבו. שיטת שמירת הנתונים צריכה לאפשר סריקה מהירה של המידע. יכולנו להגדיר לכל מקום חניה משתנה משלו, למשל parkingPlace1, parkingPlace2 וכו', אולם בדיקת כל המשתנים הייתה דורשת כתיבת תנאי בדיקה לכל משתנה בנפרד. בנייה של חניון גדול פי 10 הייתה דורשת שינוי של כל קטע התוכנית. אנו מחפשים מבנה נתונים שישמור משתנה משלו לכל מקום חניה, אבל שיאפשר גם לבדוק את כל הנתונים בלולאה, יהיה יעיל ונוח לקריאה, ויאפשר לשנות בקלות את מספר התאים שבו – כך שאותה תוכנה תתאים למספר חניונים, וההבדל יהיה רק במספר מקומות החניה שבחניון.

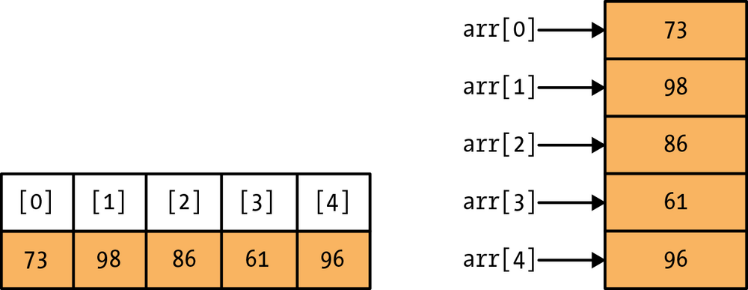
כאמור, מבנה נתונים העונה על דרישות אלו נקרא מערך (array).

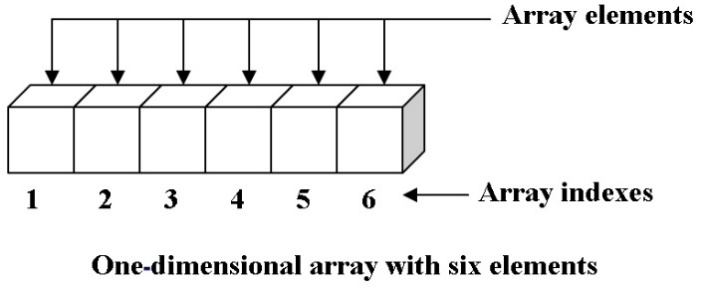
מערך הוא אוסף סדור של תאים מאותו טיפוס. לכל תא במערך אפשר להתייחס כמו למשתנה, כלומר לשמור בו ערך בהתאם לטיפוסו ולקרוא ממנו ערך. לכל תא במערך יש מספר, ומיקום של תא במערך נקרא אינדקס (מציין).

ננסה להמחיש את רעיון המערך בעזרת כמה דוגמאות מחיי היום-יום:



שימו לב שבכל התמונות יש מספר סודר על כל אחד מהתאים. מספר זה הוא האינדקס. בשלב זה לא נעמיק בשאלה כיצד נשמרים הנתונים במערך הממוחשב, ונציין רק שבניגוד לתמונות, אינדקס המיקום שלו מתחיל במספר 0.

לפניכם מספר איורים המציגים מערך כפי שהוא מיוצג במחשב:



המערך הוא מבנה נתונים חשוב כי הוא מאפשר להגיע לכל תא ישירות בציון מספרו, וכן משום שקל לשנות את מספר התאים בו והוא מאפשר לארגן את הנתונים בצורה יעילה.

נתחיל בשתי משימות פשוטות, שבשתיהן נקלוט מספרים למערך ונדפיס אותם לבדיקת הקלט. בהמשך נבצע בדיקה על המערך.



משימה 2: הגדרת מערך וקליטת נתונים לתוכו

קלט: 12 מספרים במערך.

פלט: המספר המקסימלי שנקלט.

נתחיל בהגדרה של מערך בשם myNumbers, בגודל 12, של מספרים שלמים.

הגדרה של מערך:

טיפוס המערך

הסוגריים הריבועיים מיצגים שמדובר במערך

int[] myNumbers;

שם המערך

myNumbers = new int [12];

מספר התאים במערך

הגדרת עצם חדש

המערך הוא עצם, ובדומה לעצמים שהגדרנו בעבר, גם לו יש תכונה אחת – האורך שלו. ניתן להגיע לערך זה בהוראה myNumbers.length. אך בשונה מעצמים אחרים שהכרנו, למערך אין פעולות עצמיות.

לאחר שהוגדר מערך, מה יש בו? ראינו שכאשר הגדרנו משתנה לא הושם בו ערך התחלתי והיה עלינו לאתחל אותו. בשפת ג'אווה, כאשר מגדירים אורך במערך שלמים, מושם בכל התאים הערך ההתחלתי 0.

לאחר ההגדרה נוכל לבצע על המערך את כל ההוראות שנוכל לבצע על משתנה רגיל באמצעות גישה למקום מוגדר בו.

למשל הוראת ההשמהmyNumbers[1]=9205264; , או הוספת 1 על ידי myNumbers[3]++;.

חשוב לציין שהמיקום הראשון במערך הוא המקום ה-0. כלומר במערך בגודל 10 יהיו 10 תאים הממוספרים מ-0 עד 9. דיון בשאלה מדוע המספרים מתחילים מהספרה 0 יוצג בסוף הפרק.

היתרון הגדול במערך הוא שהמשתנה אינו עומד בפני עצמו – הוא תא מספר 1 במערך של שלמים, ולכן פעולות כמו הדפסת המערך יוכלו להתבצע בלולאה. למשל, השמת ערך 1 בכל תאי המערך תיראה כך:

for (int place = 0; place < 10; place++) {

myNumbers[place] = 1;

}

נחזור למשימה. עלינו לקלוט 12 מספרים למערך:

Scanner reader = new Scanner(System.in);

for (int place = 0; place < 12; place++) {

System.out.println(“Please enter next number”);

myNumbers[place] = reader.nextInt();

}

C#

for (int place = 0; place < 12; place++) {

Console.WriteLine(“Please enter next number”);

myNumbers[place] = int.Parse(Console.ReadLine());

}

מימוש זה נוח יותר מהגדרת 12 משתנים וכתיבה של הוראות קלט לכל משתנה. הנוחות תבוא לידי ביטוי במיוחד כשנרצה לשנות את מספר התאים במערך: יספיק במקרה זה לשנות את הגדרת אורך המערך, ולא נצטרך לשנות את הלולאות עצמן.

בדומה לכך נוכל להדפיס את כל איברי המערך, אבל נבצע שינוי קטן ונשתמש בתכונה של אורך המערך. במקום לרשום 12, נשתמש ב- myNumbers.length;.

**שימו לב**, זהו יותר משינוי קוסמטי: בצורה זו נוכל להשתמש באותו קטע קוד לבדיקת כל מערך שהוא:

for (int place = 0; place < myNumbers.length; place++) {

System.out.println(“The number at place“ + place + ”is“ + myNumbers[place]);

}

C#

for (int place = 0; place < myNumbers.Length; place++) {

Console.WriteLine(“The number at place“ + place + ”is“ + myNumbers[place]);

}

**שימו לב** שאם ננסה להדפיס את כל המערך בפקודה אחת, נקבל פלט לא ברור. כדי להדפיס את ערכי המערך יש לסרוק את כל התאים בו ולהדפיס כל תא בנפרד.

כעת נותר לנו למצוא את המספר המקסימלי במערך ולהדפיס אותו. כבר מצאתם בעבר מספר מקסימלי מבין קבוצת מספרים, פעולה זו דומה גם במערך.

נגדיר משתנה שבו נשמר המספר המקסימלי, נשווה אותו לערכים במערך, ונעדכן אותו אם צריך. הערך ההתחלתי השמור במשתנה הוא הערך שבתא הראשון במערך.

המימוש:

int max = myNumbers[0];

for (int i = 0; i < myNumbers.length; i++) {

if (myNumbers[i] > max) max = myNumbers[i];

}

System.out.println(“The max number was“ + max);

C#

int max = myNumbers[0];

for (int i = 0; i < myNumbers.Length; i++) {

if (myNumbers[i] > max) max = myNumbers[i];

}

Console.WriteLine(“The max number was“ + max);

**שימו לב,** השתמשנו ב-i כאינדקס (index) הלולאה. זהו סימון נפוץ ללולאות for בכלל ובמערכים בפרט.

**רגע חושבים!**

כמה שורות היו נדרשות אם היה עלינו להשוות בין 12 מספרים ששמורים במשתנים בדידים?

ייתכן שתשאלו מדוע בכלל צריך לשמור את הנתונים שנקלטים – למה אי אפשר למצוא את הערך המקסימלי תוך כדי הקלט? זה נכון, במקרה זה הדבר אפשרי, אבל לפעמים נזדקק לבדיקה שתדרוש לשמור את כל המספרים.



### משימה 3: הגדרת מערך וקליטת נתונים לתוכו

בשאלה זו תידרשו למצוא את גודלה של קבוצת מספרים במערך המקיימת תנאי מסוים.

**קלט** 12 מספרים שלמים חיוביים.

**פלט**: כמה מהם גדולים מהמספר האחרון.

בשאלה מסוג זה לא נוכל להתחיל לבדוק לפני שנקלוט את כל המספרים.

**פתרון:**

בדומה למשימה הקודמת, עלינו לקלוט 12 מספרים למערך.

Scanner reader = new Scanner(System.in);

int[] myNumbers;

myNumbers = new int[12];

for (int place = 0; place < myNumbers.length; place++) {

System.out.println(“Please enter next number”);

myNumbers[place] = reader.nextInt();

}

C#

int[] myNumbers;

myNumbers = new int[12];

for (int place = 0; place < myNumbers.Length; place++) {

Console.WriteLine(“Please enter next number”);

myNumbers[place] = int.Parse(Console.ReadLine());

}

לאחר מכן נדפיס את כל איברי המערך:

for (int place = 0; place < myNumbers.length; place++) {

System.out.println(“The number at place“ + place + ”is“ + myNumbers[place]);

}

C#

for (int place = 0; place < myNumbers.Length; place++) {

Console.WriteLine(“The number at place“ + place + ”is“ + myNumbers[place]);

}

כעת נותר לנו למנות כמה מספרים גדולים מהאחרון, שממוקם בתא myNumbers[myNumbers.length-1]

המימוש:

int counter = 0;

for (int place = 0; place < myNumbers.length; place++) {

if (myNumbers[place] > myNumbers[myNumbers.length - 1]) counter++;

}

System.out.println("There were" + counter + "numbers, greater than the last place value");

C#

int counter = 0;

for (int place = 0; place < myNumbers.Length; place++) {

if (myNumbers[place] > myNumbers[myNumbers.Length - 1]) counter++;

}

Console.WriteLine("There were" + counter + "numbers, greater than the last place value");

Console.WriteLine("There were" + counter + "numbers, greater than the last place value");

}

Console.WriteLine(“The max number was“ + max);



### משימה 4: בדיקה האם ערך מסוים מופיע במערך

מפתחי תוכנת הניהול של החניון האוטומטי מבקשים את עזרתכם בכתיבת תוכנה שתבדוק אם מספר רכב מסוים נמצא בחניון. פרטי המכוניות שבחניון נשמרו במערך parkingPlaces. תפקידכם לכתוב קטע קוד שמפתחי התוכנה יוסיפו לתוכנית שלהם, אשר יקלוט את מספר הרכב לבדיקה וידפיס את מספר התא שבו הוא חונה. אם הרכב לא נמצא בחניון, תודפס הודעה מתאימה.

**פתחו** את האלגוריתם המבוקש:

**קלט:** מספר רכב.

**פלט:** אם הרכב חונה בחניון, תודפס הודעה עם מספר התא שבו הוא חונה. אם הוא אינו חונה בחניון, תודפס הודעה מתאימה.

**ניתוח הבעיה**

לרשותכם מערך אשר בתאיו השונים שמורים מספרים שלמים המייצגים מספרי רכב. האלגוריתם קולט מספר שלם המייצג מספר רכב. הפלט יהיה הודעה אם הרכב נמצא בחניון, ואם כן באיזה תא (בהנחה שאין מספרי רכבים זהים בתאים השונים, כפי שהוגדר בשאלה).

יש שתי דרכים אפשריות לבנות אלגוריתם כזה:

1. לסרוק את כל המערך.
2. לסרוק את המערך עד שמוצאים את הערך הרצוי.

**פתרון:**

1. נסרוק את תאי המערך עד שנמצא את מספר הרכב הרצוי. כשנמצא אותו נפסיק לחפש ונדפיס את המיקום שלו. אם סרקנו את כל התאים והמספר לא נמצא, נדפיס הודעה מתאימה.
2. נגדיר את מקומו של הרכב כמספר התחלתי של תא שאיננו קיים בחניון. כעת נסרוק את כל אברי המערך אחד אחרי השני. אם נמצא את מספר הרכב, נשמור את מספר התא שלו. אם בסיום הבדיקה מספר התא שונה ממספרו בתחילה, אז נמצא המיקום של מספר הרכב. אם לא, מספר הרכב אינו במערך.

**רגע חושבים!**

איזו לולאה מתאימה לכל אחד משני הפתרונות?

למימוש פתרון 1, שהוא בדיקת כל התאים עד למציאת המיקום הרצוי, נשתמש בלולאת **while**. משתנה בוליאני מאפשר להפסיק את הבדיקה ברגע שמוצאים את המספר הרצוי. לאחר סיום הלולאה, נבדוק האם מספר הרכב נמצא או שהסתיימה הסריקה מבלי למצוא אותו.

System.out.println("Please enter a valid user car number");

int userNumber = int.nextInt();

int place = 0;

boolean found = false;

while (!found && place < parkingPlace.length) {

if (userNumber == parkingPlace[place]) found = true;

else place++;

}

if (!found)

System.out.println("The number you entered does not exist in our parking");

else

System.out.println("Your car is stored at parking" + place);

C#

Console.WriteLine("Please enter a valid user car number");

int userNumber = int.Parse(Console.ReadLine());

int place = 0;

bool found = false;

while (!found && place < parkingPlace.Length) {

if (userNumber == parkingPlace[place]) found = true;

else place++;

}

if (!found)

Console.WriteLine("The number you entered does not exist in our parking");

else

Console.WriteLine("Your car is stored at parking" + place);

למימוש פתרון 2, שהוא בדיקת כל התאים, נשתמש בלולאת **for**. אם נמצא את המספר הרצוי, נשמור אותו במשתנה שהוגדר מראש. במשתנה זה הכנסנו מראש מספר שאינו יכול להיות מיקום במערך, למשל 1-. כך שאם לא מצאנו את מספר הרכב בין תאי המערך, המשתנה לא מתעדכן ואנו יודעים בסיום הלולאה שהמספר לא קיים במערך.

System.out.println("Please enter a valid user number");

int userNumber = int.nextInt();

int place = -1;

for (int i = 0; i < parkingPlace.length; i++) {

if (userNumber == parkingPlace[i])

place = i;

}

if (place == -1)

System.out.println("The number you entered does not exist in our parking");

else

System.out.println("Your car is stored at place number: " + place);

C#

Console.WriteLine("Please enter a valid user number");

int userNumber = int.Parse(Console.ReadLine());

int place = -1;

for (int i = 0; i < parkingPlace.Length; i++) {

if (userNumber == parkingPlace[i])

place = i;

}

if (place == -1)

Console.WriteLine("The number you entered does not exist in our parking");

else

Console.WriteLine("Your car is stored at place number: " + place);

חשוב לזכור

ניתן להחליף את התנאי בשורת הבקרה של הלולאה בתנאי מורכב שיאפשר את סיום הלולאה כשנמצא את המספר:

for (int i = 0; i < parkingPlace.length && place != -1; i++)

### עבודה עם מערך של עצמים

המערכים יכולים כמובן להיות שימושיים בהקשרים רבים נוספים מלבד תוכנית רישום חנייה. לדוגמה נוספת חפשו אתר אינטרנט שמאפשר לרכוש טלפונים סלולריים והתבוננו באופן פעולתו. ברוב האתרים תוכלו למיין את הטלפונים לפי עלות, לפי מותג ולפי פרטים נוספים, וכך למצוא את הטלפון הסלולרי שתרצו לרכוש באמצעות סקר שוק מתוחכם. לדוגמה נוכל לסנן מכשירים בטווח מחיר מסוים, או עם גודל מסך מסוים:

A screenshot of a cell phone

Description generated with very high confidenceA screen shot of a computer

Description generated with high confidenceA screenshot of a cell phone

Description generated with very high confidence

**רגע חושבים!**

כיצד ניתן לארגן את כל הנתונים על הטלפונים באמצעות מערכים?

במשימה הבאה תתנסו בכתיבת תוכנית שיודעת לקלוט נתונים על טלפונים ולבצע חלק מהאפשרויות האלה.



### משימה 5: חנות אינטרנט

חנות האינטרנט "סמארטפון\_4\_U" מעוניינת לשכור את שירותיכם בבניית תוכנית אשר מציגה למשתמש מידע על דגמי טלפונים והמחירים שלהם. המערכת מיועדת לשני משתמשים שונים. – הראשון הוא עובד החנות אשר מזין מידע על מחירי הטלפונים לפי הדגמים, והשני הוא לקוח המעוניין לקבל מידע על הטלפונים שהוא יכול לרכוש בתקציב שלו.

#### חלק א':

**פתחו** את חלק התוכנית הנדרש עבור העובד בחנות.

**קלט**: דגמים ומחירים של טלפונים סלולריים.

**הביצוע**: שמירה של כל הדגמים והמחירים של הטלפונים הסלולריים.

הנחה: הנתונים שמוקלדים אינם שגויים, כלומר לא צריך לבדוק את תקינות הקלט.

ניתוח הבעיה: עלינו לקלוט שני נתונים לכל טלפון – שם הדגם והמחיר. לכן התוכנית תקלוט שני סוגים של משתנים – קלט מסוג מספר שלם (int) למחיר, וקלט מסוג מחרוזת (String) עבור דגם. חשוב להדגיש שלאחר שקולטים את מחיר הטלפון המבוקש, הוא התקציב של הקונה, צריך לבדוק את נתוני הטלפונים שוב ולכן חייבים לשמור עליהם במבנה נתונים נוח. בלתי אפשרי לבצע את הבדיקה הזו תוך כדי קליטתם.

לפני שנראה פתרונות אפשריים, נציג את המחלקה **טלפון חכם**. **CellularPhone** העומדת לרשותכם בפתרון הבעיה.

במחלקה זו יש רק שתי תכונות של הטלפון: הדגם והמחיר.

CellularPhone

price – ערך מטיפוס שלם המייצג את מחיר הטלפון

model – ערך מטיפוס מחרוזת המייצג את דגם הטלפון

ניתן לשלוף את ערכי התכונות באמצעות הפעולות getPrice() ו- getModel(). תחילה יש להגדיר מערך של עצמים, ולאחר מכן להתייחס לכל תא במערך כאל עצם.

**פתרון**

ישנם שני פתרונות אפשריים:

1. בניית שני מערכים מתואמים לשמירת הנתונים: מערך אחד כולל מספרים שלמים המייצגים את מחירי הטלפונים, ומערך שני כולל מחרוזות המייצגות את הדגם. על מנת ליצור תיאום בין המערכים, המיקומים (האינדקסים) של כל תא בכל אחד מהמערכים צריכים להיות מתואמים – כלומר, המחיר שמופיע בתא מספר 1 במערך השלמים יהיה שייך לדגם שמופיע בתא מספר 1 במערך המחרוזות.
2. יצירת עצם מסוג טלפון סלולרי הכולל שתי תכונות, ובניית מערך של עצמים שאותו ניתן לבדוק.

נתמקד כאן בפתרון 2. כבר עבדנו עם עצמים בעבר, ונותר לנו להבין כיצד משתמשים בהם בתוך מערך.

### מערך של עצמים

בהתאם למשימה, עליכם לכתוב תוכנית המגדירה מערך של שישה טלפונים סלולריים. התוכנית קולטת עבור כל אחד מהטלפונים מחרוזת המייצגת את דגם הטלפון, ומספר שלם המייצג את מחיר הטלפון. לאחר קליטת הנתונים יש ליצור עצם ולהכניסו למערך. התוכנית קולטת מהמשתמשים מספר שלם שהוא התקציב לרכישת הטלפון, ומציגה את הדגם והמחיר של כל אחד מהטלפונים שמחירם שווה למחיר הרצוי או נמוך ממנו.

כאמור, לרשותכם עומדת המחלקה CellularPhone וניתן ליצור עצם עם שתי התכונות – דגם ומחיר. כמו כן ניתן לשלוף את ערכי התכונות על ידי פעולות getPrice() ו -getModel(). כעת עלינו להגדיר מערך של עצמים במקום מערך של מספרים שלמים, ולאחר מכן להתייחס לכל תא במערך כאל עצם.

**שימו לב!** כאשר מגדירים את המערך, התאים בו לא מכילים עצם מסוג CellularPhone, לכן צריך לשייך לכל תא עצם. כלומר עלינו להשתמש בקלט ליצירת עצמים בתוך המערך.

כמו במקרה של המספרים השלמים, גם כאן ניעזר בלולאה לבניית המערך.

תחילה נגדיר מערך של מכשירים סלולריים ונקצה עבורם שישה תאים:

CellularPhone[] myPhones = new CellularPhone[6];

טיפוס המערך

ההגדרה דומה להגדרות שביצענו קודם עם מספרים, רק טיפוס המערך משתנה.

בשלב הזה אין עדיין טלפונים סלולריים במערך. ההשמה במערך מתבצעת בשני שלבים:

1. הגדרת עצם חדש מסוג טלפון סלולרי, כולל התכונות שלו.
2. השמת העצם החדש לתוך תא המערך. ההשמה נעשית באמצעות גישה למקום המוגדר במערך.

ההוראה הבאה מגדירה עצם, אלא שהפעם הוא אינו עומד בפני עצמו אלא הוא תא מספר 0 במערך של עצמים:

myPhones[0] = new CellularPhone(phonesModel, phonesPrice);

למעשה הוראה זו זהה לכל הוראה ליצירת עצם שהכרנו קודם, אך היתרון של עצם בתוך מערך הוא היכולת להשתמש בלולאות. קטע הקוד הבא פועל באותו העיקרון כדי להכניס ערכים לכל התאים במערך באמצעות לולאה:

for (int i = 0; i < myPhones.length; i++) {

System.out.println("Please enter the model of phone number " + i);

phonesModel = reader.next();

System.out.println("Please enter the price for phone number " + i);

phonesPrice = reader.nextInt();

myPhones[i] = new CellularPhone(phonesModel, phonesPrice);

}

C#

for (int i = 0; i < myPhones.Length; i++) {

Console.WriteLine("Please enter the model of phone number " + i);

phonesModel = Console.ReadLine();

Console.WriteLine("Please enter the price for phone number " + i);

phonesPrice = int.Parse(Console.ReadLine());

myPhones[i] = new CellularPhone(phonesModel, phonesPrice);

}

בסיום הלולאה יהיו במערך נתוני הטלפונים כפי שנקלטו מהמשתמשים.



משימה 5: חנות אינטרנט

#### חלק ב':

**פתחו** את האלגוריתם הנדרש עבור הלקוח בחנות האינטרנט.

**קלט**: מספר שלם המייצג את התקציב של הלקוח, כלומר המחיר המקסימלי שהוא מוכן לשלם.

**פלט**: כל הדגמים ומחירי הדגמים מתוך המערך שמחירם שווה למספר שנקלט או נמוך ממנו.

**פתרון :**

יש להגדיר משתנה שיקלוט את הערך המקסימלי לתצוגה. זהו קלט של ערך מטיפוס שלם, ולכן נגדיר:

System.out.println("Please enter the maximum price for your phone");

int maxPrice = reader.nextInt();

C#

Console.WriteLine("Please enter the maximum price for your phone");

int maxPrice = int.Parse(Console.ReadLine());

גם את הבדיקה נבצע בלולאה:

for (int i = 0; i < myPhones.length; i++) {

if (myPhones[i].getPrice() <= maxPrice) {

numberOfPhones++;

System.out.println("Phone number " + i + " is good for you. ");

System.out.print("It’s a " + myPhones[i].getPhonesMmodel() + ", and");

System.out.println("its price is: " + myPhones[i].getPrice());

}

}

C#

for (int i = 0; i < myPhones.Length; i++) {

if (myPhones[i].getPrice() <= maxPrice) {

numberOfPhones++;

Console.WriteLine("Phone number " + i + " is good for you. ");

Console.Write("It’s a " + myPhones[i].GetPhonesMmodel() + ", and");

Console.WriteLine("its price is: " + myPhones[i].GetPrice());

}

}

### תחרות צבים

בפרקים הקודמים הכרנו את הצב, ואף יצרנו צבים שהתחרו זה בזה. חברת תוכנה המספקת שירותי אנימציה לעולם הקולנוע מתחרה כעת במכרז לכתיבת תוכנית הכוללת מערך של שמונה צבים שמתחרים זה בזה. בניגוד לתחרויות רגילות שבהן הצב שמגיע ראשון לקו הגמר הוא המנצח, כאן המנצח הוא זה שמצליח לעבור את המרחק הגדול ביותר ב-20 מהלכים, כשבכל מהלך הוא יכול להתקדם בין 0 ל-100 צעדים. הצבים מתקדמים בעזרת הגרלה של מספרים אקראיים שלמים, ובסיום מוכרז המנצח.

בשלב זה צריך להיות כבר ברור היתרון של הגדרת מערך צבים: האפשרות להשתמש בשמונה צבים שלא במסגרת מערך תהיה מאוד לא נוחה, ואם בהמשך נתבקש לעבור ל-100 צבים, מספר השינויים יהיה בלתי סביר בגודלו; לעומת זאת, נוכל לבדוק בקלות את המערך באמצעות שינוי של מספר אחד בלבד בתוכנית כולה.

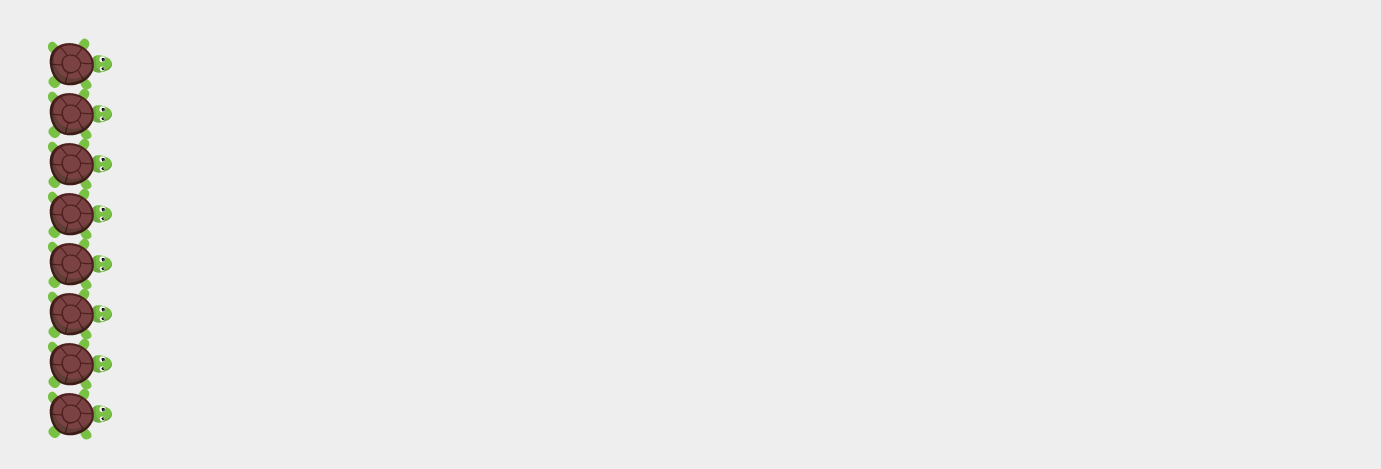
בתכנון התוכנית הוגדרו שלושה שלבים:

1. הגדרת מערך הצבים והתמקמות בקו הזינוק.
2. התחרות עצמה.
3. הכרזה המנצח.

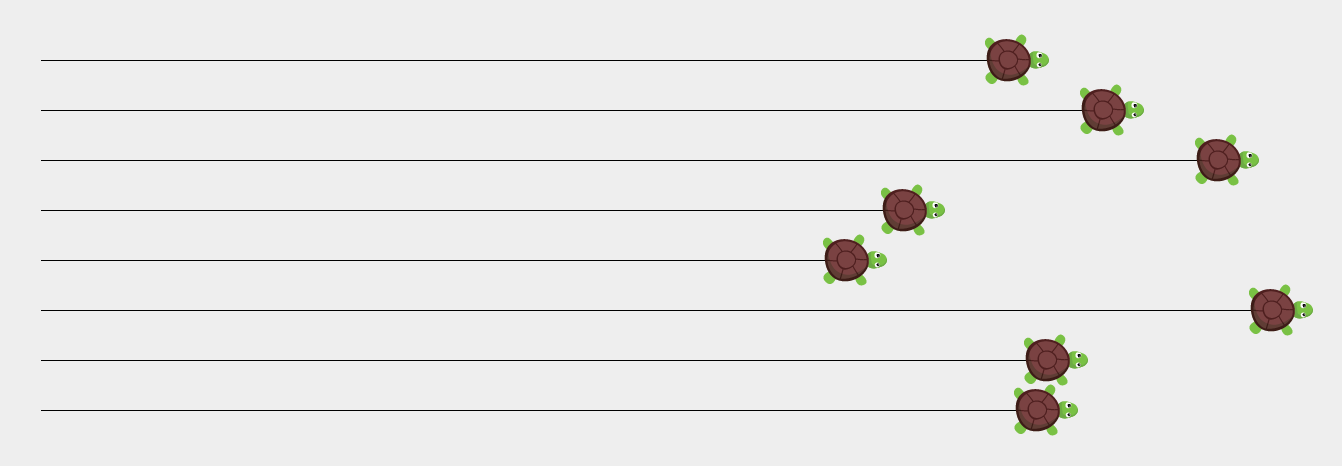
שלבים 1 ו-2 כבר תוכנתו ועובדים היטב, ותוכלו להעתיק אותם ולבדוק זאת. אולם את השלב השלישי לא הספיקו המתכנתים לסיים. המשימה שלכם היא לעזור להם לסיים את התוכנית.

**דוגמה למרוץ**

התמקמות בקו הזינוק:



תוצאת המרוץ:



מימוש שני השלבים הראשונים:

הגדרת מערך הצבים ומיקומם על קו הזינוק מצד שמאל של המסך:

שימו לב להתקדמות בהתאם למיקום במערך:

Turtle[] turtles = new Turtle[8];

for (int i = 0; i < turtles.length; i++) {

turtles[i] = new Turtle();

turtles[i].moveForward(i \* 50);

turtles[i].turnRight(90);

turtles[i].moveBackward(800);

turtles[i].tailDown();

}

C#

Turtle[] turtles = new Turtle[8];

for (int i = 0; i < turtles.Length; i++) {

turtles[i] = new Turtle();

turtles[i].MoveForward(i \* 50);

turtles[i].TurnRight(90);

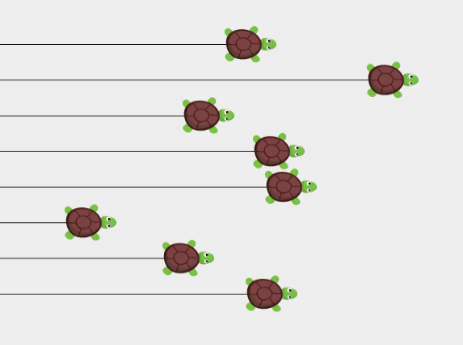
turtles[i].MoveBackward(800);

turtles[i].TailDown();

}

המרוץ עצמו:

שימו לב שיש כאן שתי לולאות מקוננות, אחת רק לחזרה והשנייה עוברת בין הצבים. כמו כן הוגדרה השהיה כדי שניתן יהיה לראות את ההתקדמות בהדרגה, ולא שנראה בבת אחת את הצבים בקו הסיום.



for (int round = 1; round <= 20; round++) {

for (int i = 0; i < turtles.length; i++) {

turtles[i].moveForward((int)(Math.random() \* 101));

turtles[i].setDelay(50);

}

}

C#

Random rnd = new Random();  
for (int round = 1; round <= 20; round++) {  
 for (int i = 0; i < turtles.Length; i++) {  
 turtles[i].MoveForward(rnd.Next(101)+1);  
 }

}

תוכלו לצפות בסרטון של מרוץ לדוגמה בקישור הבא: [מרוץ צבים](https://drive.google.com/file/d/1jRTVk_apNYtMC6LOPYpk7kewT0yI5reO/view?usp=sharing)



### משימה 6: גישה לתכונה של עצם במערך

הצב המנצח הוא זה שהתקדם במספר הצעדים הגדול ביותר, כלומר המרוחק ביותר, מקו הזינוק.

אחת מתכונות הצב היא מיקומו בציר ה-x ואותה נחפש במרוץ זה. ניתן להגיע אליה בעזרת הפעולה

turtles[i].getXLocation()

שימו לב שלצב שלנו אין שם, זהו הצב שממוקם בתא i במערך הצבים שלנו.

המימוש:

int bestPlace = 0;

for (int i = 0; i < turtles.length; i++) {

if (turtles[bestPlace].getXLocation() < turtles[i].getXLocation())

bestPlace = i;

}

System.out.println("the best turtle was the one on track " + i);

C#

int bestPlace = 0;

for (int i = 0; i < turtles.Length; i++) {

if (turtles[bestPlace].GetXLocation() < turtles[i].GetXLocation())

bestPlace = i;

}

Console.WriteLine("the best turtle was the one on track " + i);

כמה הערות:

בהתחלה הוגדר הצב במקום ה-0 כצב המנצח, אבל זה לא באמת משנה כי אם יהיה צב טוב ממנו המיקום המנצח יעודכן. אם לא יהיה צב טוב ממנו, הוא יישאר המנצח.

מה קורה אם יש שני צבים שהגיעו לאותו מרחק בסיכום המהלכים? מבחינת התוכנה יוגדר כמנצח הצב שלתא שלו יש מספר קטן יותר (חִשבו למה). ניתן לפתור זאת במציאת המרחק המנצח, ולאחר שמצאנו אותו לסרוק שוב את המערך ולהדפיס את המקומות של כל הצבים שתוצאתם שווה לשלו.

מימוש:

for (int i = 0; i < turtles.length; i++) {

if (turtles[bestPlace].getXLocation() < turtles[i].getXLocation())

System.out.println("The best turtle was the one on track " + i);

}

C#

for (int i = 0; i < turtles.Length; i++) {

if (turtles[bestPlace].GetXLocation() < turtles[i].GetXLocation())

Console.WriteLine("The best turtle was the one on track " + i);

}



הוסיפו קטע קוד שבו כל הצבים המנצחים מבצעים סיבוב ניצחון כמו בסרטון.

### פעולות עזר ומערכים

בפרקים הקודמים למדנו שהתוכנית הראשית הופכת להיות ארוכה ומסורבלת אם לא משתמשים בפעולות עזר. כמו כן, למדנו כיצד לשלוח ערך כפרמטר לפעולת עזר.

למשל כותרת הפעולה שמקבלת שני מספרים שלמים ומחזירה את הערך של הגדול מבין השניים תיראה כך:

public static int maxOf2(int number1, int number2)

שם הפרמטר

טיפוס הפרמטר

טיפוס הערך המוחזר

**רגע חושבים!**

אם עומדים לרשותכם 300 מספרים שלמים וברצונכם למצוא את הערך הגבוה ביותר מביניהם, כיצד תוכלו להיעזר במערך ובפעולת העזר למציאתו?

נוכל למצוא את הערך המקסימלי במערך באמצעות פעולה דומה לפעולת העזר לעיל, אך הפעם הפעולה פועלת על מערך:

public static int maxOfArray(int [ ] numbersArray)

שם המערך

טיפוס המערך



### משימה 7: שינוי ערכי מערך בפעולת עזר

המורים למדעי המחשב החליטו לקראת הבגרות לכתוב אלגוריתם שישפר אוטומטית את הציונים של תלמידי הכיתה המצטיינים. האלגוריתם יעגל ל-95 את הציונים שגבוהים מ-90 ונמוכים מ-95, ויעגל ל-100 את כל הציונים שגבוהים מ-95.

הציונים שמורים במערך של מספרים ממשיים, שכן המורים אוהבים לתת ציונים שאינם שלמים – כמו 91.5.

כמו כן האלגוריתם יחזיר את מספר התלמידים שציונים קטן או שווה ל-90 כדי לאפשר להם להיבחן במבחן חוזר.

**כתבו** פעולה שמקבלת מערך של מספרים ממשיים, מתקנת את הציונים בהתאם למתואר ומחזירה את מספר הציונים במערך שהם מ-90 ומטה.

**פתרון**

ניתוח הבעיה:

בשונה מהמשימות שעשיתם קודם לכן, הפעם עליכם לכתוב פעולה שמקבלת מערך כפרמטר, לשנות את ערך המערך, ולשמור את השינוי גם בתוכנית הראשית. הסיבה לכך היא שהנתונים שהועברו מצביעים בעצם אל המערך שכבר הוגדר. כלומר יש כאן הפניה אל המערך המקורי שהוגדר מחוץ לפעולה. איננו משכפלים את המערך עצמו אלא רק את ההפניה למקום שכבר קיים בזיכרון. המשמעות היא שאם נשנה משהו במערך תוך כדי השימוש בו בפעולה, בעצם נשנה את המערך המקורי.

למעשה נתכנן את הפתרון כמו קודם: ראשית נגדיר מונה שימנה כמה מספרים גבוהים מ-90, ונאתחל אותו ל-0. לאחר קבלת המערך נסרוק את הנתונים ונשווה תחילה כל נתון ל-90. אם הערך גבוה מכך – נעדכן את המונה, ואז נשווה את הערך גם ל-95 ונעדכן במקרה הצורך.

לאחר בדיקת כל המערך יוחזר ערך המונה לפעולה שזימנה פעולה זו.

פתרון אפשרי:

public static int updateGradesInArray(double[] numbersArray) {

int counter = 0;

for (int i = 0; i < numbersArray.length; i++) {

if (numbersArray[i] > 90) {

if (numbersArray[i] > 95)

numbersArray[i] = 100;

else

numbersArray[i] = 95;

}

else

counter++;

}

return counter;

}

C#

public static int UpdateGradesInArray(double[] numbersArray) {

int counter = 0;

for (int i = 0; i < numbersArray.Length; i++) {

if (numbersArray[i] > 90) {

if (numbersArray[i] > 95)

numbersArray[i] = 100;

else

numbersArray[i] = 95;

}

else

counter++;

}

return counter;

}



### משימה 8: הפקטור

לאחר מבחן קשה במיוחד, החליט המורה לבצע פקטור שורש בציוני התלמידים. הפקטור מבצע שתי פעולות מתמטיות: תחילה מחשבים את שורש הציון המקורי, ולאחר מכן מכפילים את התוצאה ב-10. בשיטה הזו מי שקיבל לדוגמה 30 יקבל בסוף 55, ומי שקיבל 100 יקבל עדיין 100.

**כתבו** פעולה שמקבלת מערך של מספרים ומתקנת את הציונים בו בעזרת פקטור שורש.

**פתרון :**

ניתוח הבעיה:

לאחר קבלת המערך יש לסרוק את ערכיו ולבצע את הפעולה על כל אחד מהם. בנוסף יש לזכור שאנו עובדים עם שלמים. שימו לב כי בסיום הפעולה אין צורך להחזיר (return) את המערך, שכן הפעולה מתבצעת על המערך המקורי מלכתחילה.

מימוש פתרון אפשרי:

public static void multiplyArrayByNumber(int[] grades){

הפעולה אינה מחזירה ערך כלשהו

for (int i = 0; i < grades.length; i++) {

grades[i] = (int)(10 \* Math.sqrt(grades[i]));

}

}

### 

### משימה 9: ממיר דולרים לשקלים

אחת הפעולות הנפוצות של תוכניות מחשב היא המרת ערכים – למשל, המרת ערך מטבע מדולרים לשקלים.

בטבלה הבאה נוכל לראות המרה מדולרים לשקלים, כאשר ערכו של דולר אחד שווה 3.9 שקלים.

|  |  |  |
| --- | --- | --- |
| **דולרים** | **ערך הדולר** | **שקלים** |
| 1.0 |  | 3.9 |
| 2.0 |  | 7.8 |
| 3.0 |  | 11.7 |
| 4.0 |  | 15.6 |
| 5.0 |  | 19.5 |

כיוון שערך הדולר משתנה, התוכנית צריכה להציג את ההמרה בהתאם לערך נתון.

**כתבו** פעולה שמקבלת שני מערכים של מספרים ממשיים ומספר ממשי נוסף. המערך הראשון מכיל מספרים שמייצגים ערך בדולרים, והמערך השני מכיל אפסים. המספר הנוסף הוא ערך הדולר. הפעולה תציב במערך השני את הערך השקלי של כל אחד מהערכים במערך הראשון באותו מיקום.

למשל למערך dollars = {1.0,2.0,3.0,4.0,5.0} ולערך הדולר value = 3.9 ולזימון:

fromDollarsToShekels(dollars, shekels, value);

נקבל shekels = {3.9,7.8,11.7,15.6,19.5}

**פתרון**

ניתוח הבעיה:

לאחר קבלת המערך עלינו לסרוק אותו ולבצע את הפעולה על כל ערך בתוכו. יש לזכור שאנו עובדים עם מספרים ממשיים. שימו לב כי בסיום משתנים ערכי המערך shekels, שכן הפעולה מתבצעת על המערך המקורי.

מימוש פתרון אפשרי:

public static void fromDollarsToShekels(double[] dollars, double[] shekels, double value); {

for (int i = 0; i < dollars.length; i++) {

shekels[i] = dollars[i] \* value;

}

}

C#

public static void FromDollarsToShekels(double[] dollars, double[] shekels, double value); {

for (int i = 0; i < dollars.Length; i++) {

shekels[i] = dollars[i] \* value;

}

}

### פעולה שמחזירה מערך

בדוגמאות הקודמות שלחנו מערכים לפעולות וקיבלנו ערכים שביקשנו, או ששינינו את המערכים בהתאם לדרישה מסוימת. לעיתים נרצה שהפעולות ייצרו מערכים בעצמן, והפעולה הראשית תקבל את המערך המוחזר שהוגדר בפעולה.



### משימה 10: משחק המספרים

לצורך משחק חברתי בערב שכבה הוחלט לחלק את המשתתפים לקבוצות בגודל אקראי. גדלי הקבוצות יישמרו בתוך מערך שיודפס וישמש את מנחה הערב. בגלל שלא ידוע כמה תלמידים יגיעו לערב, הוחלט לכתוב פעולה שתקבל את מספר התלמידים שהגיעו ותבנה מערך בהתאם.

**כתבו** פעולה שמקבלת מספר שלם ומחזירה מערך של מספרים שלמים אקראיים בין 1 ל-30 שאורכו באורך המספר.

**פתרון**

ניתוח הבעיה:

בבעיות הקודמות הפעולה קיבלה הפניה למערך מוגדר, ואילו הפעם עלינו להגדיר מערך חדש בפעולה עצמה. גודל המערך יהיה בהתאם למספר שהתקבל. לאחר מכן יש לסרוק את כל תאי המערך ולשמור בהם מספר אקראי בתחום, ובסיום הפעולה להחזיר את המערך.

החזרת מערך נועדה לכך שנוכל להשתמש בו בתוכנית הראשית. כיוון שהגדרת מערך אינה מוכרת בפעולה הראשית, כדי שהיא תוכל לקבל את המערך המוחזר יש להגדיר בה מראש מערך ללא ערכים וללא אורך. למעשה כבר עשינו זאת כשהצגנו איך להגדיר מערך:

int [ ] myArray;

בעבר ביצענו את ההשמה בשורה שלאחר ההגדרה, והפעם השמת הקריאה לפעולה במערך שיצרנו תגרום למערך שהוגדר בתוכנית הראשית להצביע על המערך שהוגדר בפעולה.

למשל, בתוכנית הראשית נגדיר את המערך randomNumbersArray, ואז נקלוט מה האורך הרצוי לו ונקרא לפעולה createRandomNumbersArray(int length):

מימוש פתרון אפשרי:

בתוכנית הראשית, בהנחה שהוגדר reader:

int[] randomNumbersArray;

int numberOfStudents = reader.nextInt();

randomNumbersArray = createRandomNumbersArray(10);

C#

int[] randomNumbersArray;

int numberOfStudents = int.Parse(Console.ReadLine());

randomNumbersArray = CreateRandomNumbersArray(10);

שימו לב כי בדוגמה זו אנו מבצעים זימון של הפעולה עם המספר הרצוי, והתוצאה המוחזרת תשויך למערך שהוגדר בפעולה הראשית.

טיפוס הערך המוחזר הוא מערך

public static int[] createRandomNumbersArray(int aNumber){

הגדרת מערך חדש

int[] tempRandomNumbersArray = new int[aNumber];

for (int i = 0; i < tempRandomNumbersArray.length; i++) {

tempRandomNumbersArray[i] = (int)(Math.random() \* 30 + 1);

}

return tempRandomNumbersArray;

}

C#

טיפוס הערך המוחזר הוא מערך

public static int[] CreateRandomNumbersArray(int aNumber){

הגדרת מערך חדש

int[] tempRandomNumbersArray = new int[aNumber];

Random rnd = new Random();

for (int i = 0; i < tempRandomNumbersArray.Length; i++) {

tempRandomNumbersArray[i] = rnd.next(30)+1;

}

return tempRandomNumbersArray;

}

בשלב זה אתם אולי כבר יודעים שניתן לממש את הגרלת המספר האקראי בצורה נוספת. אם נגדיר עצם מסוג Random (דורש גם יבוא של java.util.\*), נוכל להשתמש בפעולה:

tempRandomNumbersArray [i] = rand.nextInt(31);

C#

tempRandomNumbersArray[i] = rand.nextInt(30)+1;

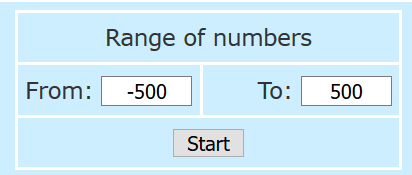
### חיפוש במערך ממוין

במשימה 4 חיפשנו ערך מסוים במערך שלא היה ממוין בהכרח. הצגנו שתי שיטות לעשות זאת, וראינו ששימוש בלולאת **while** מקצר את מספר הבדיקות – שכן לאחר שמצאנו את הערך הרצוי, אין צורך להמשיך לבדוק. בממוצע ניתן לומר שבפיזור אקראי של ערכים, מספר הבדיקות הדרוש יהיה כמחצית מאורך המערך.

חיפוש זה הוא חיפוש סדרתי. האם ניתן לשפר שיטת חיפוש זו?



### משימה 11: אלגוריתם חיפוש

**היכנסו** **ושחקו** במשחק מציאת מספר, הגדירו תחום משחק של מספרים בתחום של 500- עד 500:

<https://www.mathsisfun.com/games/guess_number.html>

**היכנסו** **ושחקו** במשחק מציאת מספר נוסף:

<http://www.abcya.com/guess_the_number.htm>

לחצו על לחצן ההתחלה  ואז על ההפעלה  ובחרו במשחק בתחום 500- עד 500

**שימו לב** שבמשחק זה יש הגבלת מהלכים.

האם הצלחתם בשני המשחקים? כיצד לדעתכם ידעו כותבי המשחק השני שניתן למצוא את המספר שהם בחרו בעזרת עשרה מהלכים לכל היותר? מה השיטה?

**פתרון**

לצורך מציאת האלגוריתם, חשבו על המספר כחלק מרצף של מספרים בעלי יחסים ביניהם – הוא קטן מחלק מהם וגדול מאחרים. באמצעות יחסים אלו אפשר לצמצם את חיפוש המספרים: למשל במקום לחפש בין 0 ל-100, נחפש בין 14 ל-57. אלגוריתם זה נקרא **חיפוש בינארי** והוא מתבסס על ההנחה שהמספרים ממוינים. בכל תור נבחר מספר שמחלק את התחום לשני חלקים – החלק שבו נמצא המספר והחלק שבו הוא לא נמצא. הבחירה במספר שבמרכז התחום מקטינה בכל ניחוש את התחום בערך פי 2 ומבטיחה שנגיע אל המספר המבוקש במהירות.

נוכל למצוא שיטה זו גם בשם "אריה במדבר". אם יש בידינו חיישן שמציין אם האריה נמצא מימין או משמאל, נוכל לבצע את החיפוש המתואר.



**רגע חושבים!**

מדוע בתחום שאורכו 1000 מספרים נוכל להגיע למספר הרצוי בתוך עשרה ניחושים לכל היותר? בהמשך נראה את החישוב המדויק לכך.

**צפו** בסרטון [חיפוש בינארי](https://www.youtube.com/watch?v=iP897Z5Nerk) המציג חיפוש דומה לשיטת ניחוש המספר, אבל במערך.

אנו מניחים כמובן שהערך הרצוי נמצא במערך (<https://www.youtube.com/watch?v=iP897Z5Nerk>)

נראה דוגמה לחיפוש במערך ממוין אשר מוצא מספר מתוך המערך באופן יעיל יותר מאשר החיפוש הסדרתי שביצענו עד כה.

האלגוריתם: נבדוק את ערך התא שנמצא באמצע המערך. אם מצאנו את הערך הרצוי, סיימנו. אם הוא גדול מהערך הרצוי, אזי הערך הרצוי ממוקם בהכרח בתא שהאינדקס שלו קטן מזה של אמצע המערך. אם הוא קטן מהערך הרצוי, אזי הערך הרצוי נמצא בהכרח בתא שהאינדקס שלו גדול מזה משל הערך הנבדק.



### משימה 11: חיפוש מיקום של ערך במערך ממוין

**כתבו** פעולה המקבלת מערך ממוין של מספרים שלמים ומספר שלם נוסף.

הפעולה תחזיר את מקומו של המספר במערך הממוין בשיטת החיפוש הבינארי בהנחה שהמספר קיים במערך.

ניתוח הבעיה:

**קלט**: הפעולה מקבלת מערך מספרים ממוין ומספר לחיפוש במערך.

**פלט**: הפעולה מחזירה את מיקום המספר במערך.

הנחה: המערך חייב להיות ממוין והערך מופיע במערך.

מימוש הפתרון במערך ממוין על ידי חיפוש בינארי:

public static int binSearch(int[] sortedArray, int number) {

int leftPlace = 0;

int rightPlace = sortedArray.length - 1;

int placeOfNumber = -1;

int placeToSearch = (leftPlace + rightPlace) / 2;

while (placeOfNumber == -1) {

\* System.out.println("first place to search" + placeToSearch);

if (sortedArray[placeToSearch] == number) {

placeOfNumber = placeToSearch;

} else {

if (sortedArray[placeToSearch] > number) {

rightPlace = placeToSearch - 1;

placeToSearch = (leftPlace + rightPlace) / 2;

} else // the value is bigger...

{

leftPlace = placeToSearch + 1;

placeToSearch = (leftPlace + rightPlace) / 2;

}

}

}

return placeOfNumber;

}

C#

public static int BinSearch(int[] sortedArray, int number) {

int leftPlace = 0;

int rightPlace = sortedArray.Length - 1;

int placeOfNumber = -1;

int placeToSearch = (leftPlace + rightPlace) / 2;

while (placeOfNumber == -1) {

\* Console.WriteLine("first place to search" + placeToSearch);

if (sortedArray[placeToSearch] == number) {

placeOfNumber = placeToSearch;

} else {

if (sortedArray[placeToSearch] > number) {

rightPlace = placeToSearch - 1;

placeToSearch = (leftPlace + rightPlace) / 2;

} else // the value is bigger...

{

leftPlace = placeToSearch + 1;

placeToSearch = (leftPlace + rightPlace) / 2;

}

}

}

return placeOfNumber;

}

**שימו לב** לשורה המסומנת ב-\*, זוהי שורת בקרה שנועדה להציג את התקדמות התוכנית.

משימה לבדיקה עצמית – הוסיפו מונה שימנה כמה פעמים מתבצעת לולאת while במערך שיש בו עשרה תאים עם הערכים: 0.1.2.3.4.5.6.7.8.9, והערך הרצוי 9. רשמו את התוצאה.

הגדילו את המערך לאורך 50, והציבו בו את המספרים 0 עד 49. כמה פעמים הלולאה מתבצעת למערך זה עבור הערך הרצוי 48?

כמה פעמים לדעתכם תתבצע הלולאה למערך המכיל את המספרים מ-0 עד 999 (1000 מקומות), והערך הרצוי הוא 0?

**רגע חושבים!**

מה היה קורה אם הערך המבוקש לא היה קיים במערך?

האם התוכנית עלולה להיכנס ל"לולאה אין סופית"? אם כן, האם ניתן לתקן אותה כדי שזה לא יקרה?



### משימה 12: היזהרו מלולאות אינסופיות

תקנו את הפעולה שכתבתם, כך שהפעם תניחו כי אם המספר לא מופיע במערך יוחזר הערך 1-.

**קלט:** הפעולה מקבלת מערך ממוין של מספרים שלמים ומספר שלם נוסף.

**פלט:** הפעולה תחזיר את מקומו של המספר במערך הממוין בשיטת החיפוש הבינארי. אם המספר לא מופיע במערך, יוחזר הערך 1-.

הנחה: המערך חייב להיות ממוין.

**פתרון**

לפני שנתחיל בפתרון, נציג כלי בקרה אפשרי נוסף כדי שנוכל לבחון מה קורה בלולאות אינסופיות.

בפתרון הקודם דרשנו את שורת הבקרה \*. בפתרון זה ניתן להוסיף אחריה שורת בקרה נוספת –

reader.nextLine();

C#

Console.ReadKey();

שורה זו תמתין לקלט לפני תחילת הבדיקה הבאה, ונוכל לראות את השלבים להיווצרות לולאה אינסופית.

הקלט לא חייב להיות מספר או תו, גם לחיצה על Enter תספיק.

שימוש בשיטה זו יגלה שכאשר המספר לא נמצא, מערך התוכנית בודק שוב ושוב מקטע בגודל תא בודד.

יש כמה דרכים לתקן בעיה זו, ואנו נבחר בשינוי ערך המשתנה שבו מוצב המיקום של המספר שחיפשנו (placeOfNumber). מטרת השינוי היא לגרום לכך שהלולאה תיפסק, ומחוץ לה נוכל לבדוק מדוע היא פסקה.

בגלל ש-1- הוא ערך בעל משמעות במערך זה, נאתחל את המשתנה האחראי על הלולאה למספר אחר, למשל 2-.

חיפוש בינארי למקרה שהערך המבוקש אינו נמצא במערך:

public static int binSearch2(int[] sortedArray, int number) {

int leftPlace = 0;

int rightPlace = sortedArray.length - 1;

int placeOfNumber = -2;

int placeToSearch = (leftPlace + rightPlace) / 2;

while (placeOfNumber == -2) {

\* System.out.println("first place to search" + placeToSearch); \*\*

System.out.println("left is " + sortedArray[leftPlace] + " right is " +

sortedArray[rightPlace]); \*\*

\* reader.nextLine();

if (sortedArray[placeToSearch] == number) {

placeOfNumber = placeToSearch;

} else {

$

if (leftPlace > rightPlace) placeOfNumber = -1;

if (sortedArray[placeToSearch] > number) {

rightPlace = placeToSearch - 1;

placeToSearch = (leftPlace + rightPlace) / 2;

} else // the value is bigger...

{

leftPlace = placeToSearch + 1;

placeToSearch = (leftPlace + rightPlace) / 2;

}

}

}

return placeOfNumber;

}

השורות המסומנות ב-\*, \*\*, \*\*\* הן שורות בקרה. בשורה הכוללת סימן $ מובן כבר שהמספר לא נמצא במערך.

C#

public static int BinSearch2(int[] sortedArray, int number) {

int leftPlace = 0;

int rightPlace = sortedArray.Length - 1;

int placeOfNumber = -2;

int placeToSearch = (leftPlace + rightPlace) / 2;

while (placeOfNumber == -2) {

\* Console.WriteLine("first place to search" + placeToSearch); \*\*

Console.WriteLine("left is " + sortedArray[leftPlace] + " right is " +

sortedArray[rightPlace]); \*\*

\* Console.ReadKey();

if (sortedArray[placeToSearch] == number) {

placeOfNumber = placeToSearch;

} else {

$

if (leftPlace > rightPlace) placeOfNumber = -1;

if (sortedArray[placeToSearch] > number) {

rightPlace = placeToSearch - 1;

placeToSearch = (leftPlace + rightPlace) / 2;

} else // the value is bigger...

{

leftPlace = placeToSearch + 1;

placeToSearch = (leftPlace + rightPlace) / 2;

}

}

}

return placeOfNumber;

}

### מיון מערך

עד כה ראינו בבירור את יתרונותיו הרבים של מערך ממוין. כאשר נשתמש במערך עבור נתונים רבים, עדיף לשמור אותם כך שהשליפה תהיה יעילה ונוחה. מערך ממוין יאפשר למצוא במהירות את הנתון שאנו מחפשים.

**רגע חושבים!**

לפניכם רשימת המספרים 5,2,3,7,8,1,6,9. כיצד נראית רשימת המספרים כאשר היא ממוינת? באילו דרכים נוכל לבצע את תהליך המיון? שאלו את חבריכם לכיתה כיצד הם ניגשו לתהליך המיון והשוו בין הגישות.

כעת נכיר שני אלגוריתמים למיון מערך. אלו אינם האלגוריתמים הידועים כיעילים ביותר, אבל הם ברורים ונוחים להבנה ולמימוש. ישנם אלגוריתמים רבים למיון, תוכלו לראות כמה מהם באתר:  
[Sorting Algorithms Animation](https://www.toptal.com/developers/sorting-algorithms)

נסו להבחין איזה אלגוריתם יעיל יותר לפי המצב ההתחלתי של המערך למיון.



משימה 13: מיון בועות

**צפו** בסרטון המציג מיון בועות ב-youtube: [bubble sort dance](https://www.youtube.com/watch?v=lyZQPjUT5B4)

**הסבירו** מהם המאפיינים הייחודיים של מיון הבועות?

כך נמיין מספרים באמצעות מיון בועות:

נשווה את הערך במקום הראשון (משמאל) לערך במקום השני. אם הערך הראשון גדול מהשני, נחליף ביניהם. אחרת נשאיר את הסדר כפי שהיה.

נשווה את הערך של המקום השני, שכעת הוא בהכרח גדול או שווה לערך של המקום הראשון, עם הערך של המקום השלישי. אם הערך השני גדול מהשלישי, נחליף ביניהם. אחרת נשאיר את הסדר כפי שהיה.

ממשיכים כך עד שמשווים את הערך הלפני אחרון עם הערך האחרון, וגם כאן מחליפים ביניהם במידת הצורך.

בשלב הזה אנו יודעים בוודאות שהערך במקום האחרון הוא הגדול ביותר מכל הערכים שבמערך.

נראה את התהליך בפועל על המערך הבא:

**6** **2** 5 9 3

בהשוואה הראשונה נחליף בין 6 ל-2 ונקבל: **2** **6** 5 9 3

בהשוואה השנייה נחליף בין 6 ל-5 ונקבל: 2 **5** **6** 9 3

בהשוואה השלישית נחליף בין 6 ל-3 ונקבל: 2 5 **6** **9** 3

בהשוואה הרביעית בין 6 ל-9 לא תתבצע החלפה כלל, ונקבל: 2 5 6 **3** **9**

באלגוריתם זה הבדיקה הראשונה אינה ממיינת את המערך כולו, אבל מבטיחה שהמספר הגדול ביותר הגיע למקומו.

אם נסרוק את כל המערך מספר פעמים כמספר התאים, בסוף התהליך המערך יהיה ממוין.

שימו לב שאין באמת צורך לבצע בכל שלב השוואות עד סוף המערך, שכן האיברים שם כבר ממוינים. אולם ההשוואות האחרונות אינן מזיקות ומשאירות את הלולאות במבנה הפשוט שלהן. בהמשך נבקש שתשנו אלגוריתם זה כך שבדיקות אלו לא יתבצעו.

במקרה שלנו אלו יהיו ההעברות:

**6** **2** 5 9 3 -> 2 **6** **5** 9 3 -> 2 5 **6** **9** 3 -> 2 5 6 **9** **3** -> 2 5 6 **3 9**

**2** **5** 6 3 9 -> 2 **5** **6** 3 9 -> 2 5 **6** **3** 9 -> 2 5 3 **6** **9** -> 2 5 3 **6** **9**

**2** **5** 3 6 9 -> 2 **5** **3** 6 9 -> 2 3 **5** **6** 9 -> 2 3 5 **6** **9** -> 2 3 5 **6** **9**

**2** **3** 5 6 9 -> 2 **3** **5** 6 9 -> 2 3 **5** **6** 9 -> 2 3 5 **6** **9** -> 2 3 5 **6** **9**

**2** **3** 5 6 9 -> 2 **3** **5** 6 9 -> 2 3 **5** **6** 9 -> 2 3 5 **6** **9** -> 2 3 5 **6** **9**

ניתן לראות שבתשע הבדיקות האחרונות לא נעשתה כל החלפה, אבל הבדיקות נדרשות לפי האלגוריתם שבנינו.

**רגע חושבים!**

נסו לשנות את הלולאות כך שלא ייערכו ההשוואות המיותרות.

**רגע חושבים!**

מה צריך להיות הסידור המקורי כדי שלא תתבצע כל החלפה? מה צריך להיות הסידור המקורי כדי שבכל בדיקה תתבצע החלפה?

תחילה נגדיר את המערך ונכניס בו ערכים שונים:

int[] bubbleSortArray = new int[9];

for (int i = 0; i < bubbleSortArray.length; i++)

bubbleSortArray[i] = bubbleSortArray.length - i;

C#

int[] bubbleSortArray = new int[9];

for (int i = 0; i < bubbleSortArray.Length; i++)

bubbleSortArray[i] = bubbleSortArray.Length - i;

נדפיס אותו לבקרה:

System.out.print("The original array: ");

for (int i = 0; i < bubbleSortArray.length; i++)

System.out.print(bubbleSortArray[i] + " ");

System.out.println();

for (int i = 0; i < bubbleSortArray.length; i++) {

for (int place = 0; place < bubbleSortArray.length - 1; place++) {

if (bubbleSortArray[place] > bubbleSortArray[place + 1]) {

כאן מתבצעת החלפה.

int temp = bubbleSortArray[place];

bubbleSortArray[place] = bubbleSortArray[place + 1];

bubbleSortArray[place + 1] = temp;

}

}

}

System.out.print("The sorted array: ");

for (int i = 0; i < bubbleSortArray.length; i++)

System.out.print(bubbleSortArray[i] + " ");

System.out.println();

C#

Console.Write("The original array: ");

for (int i = 0; i < bubbleSortArray.Length; i++)

Console.Write(bubbleSortArray[i] + " ");

Console.WriteLine("");

for (int i = 0; i < bubbleSortArray.Length; i++) {

for (int place = 0; place < bubbleSortArray.Length - 1; place++) {

if (bubbleSortArray[place] > bubbleSortArray[place + 1]) {

כאן מתבצעת החלפה.

int temp = bubbleSortArray[place];

bubbleSortArray[place] = bubbleSortArray[place + 1];

bubbleSortArray[place + 1] = temp;

}

}

}

Console.Write("The sorted array: ");

for (int i = 0; i < bubbleSortArray.Length; i++)

Console.WriteLine(bubbleSortArray[i] + " ");

Console.WriteLine("");

שימו לב שההשוואות בלולאה בוצעו כל עוד המיקום להחלפה קטן מהמיקום של הערך האחרון, כלומר:

place < bubbleSortArray.length – 1

זאת משום שלא ניתן להשוות את הערך של התא במקום האחרון לערך בתא שאחריו.

עיון באלגוריתם מגלה שבכל סריקה אנו משווים בין כל שני איברים סמוכים ומחליפים ביניהם במקרה הצורך. במקרים קיצוניים, למשל כשהמערך מסודר הפוך מהסדר הרצוי, מתבצעות החלפות רבות בכל בדיקה. ניתן לשפר את האלגוריתם בכך שנשמור את תהליך הבדיקה אבל נבצע רק החלפה אחת בכל סבב – בין האיבר שנבחר כמקסימלי לבין האיבר במיקום שבו הוא צריך להיות במערך הממוין. מיון זה נקרא מיון בחירה.

### מיון בחירה

מיון בחירה דומה למיון הקודם בכך שגם בו אנו משווים בין ערכי התאים במערך. אולם הפעם במקום להחליף בין הערכים, אנו שומרים את האינדקס של הערך הגדול מבין השניים לקראת ההשוואות הבאות. בסיום הבדיקה יהיה בידינו האינדקס של הערך בגדול ביותר, ואותו נחליף בערך במקום הכי קרוב לסוף המערך שעדיין לא מוין.

בדרך זו אומנם לא נחסוך בהשוואות, אבל כן נחסוך בהחלפות.

כיצד ניגש לבצע מיון בועות במערך בשיטה זו?

תחילה הערך המקסימלי מבחינתנו נמצא בתא הראשון, שהוא תא מספר 0.

נרשום זאת כך: maxPlace=0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0** | **1** | 2 | 3 | 4 |
| 6 | 2 | 5 | 9 | 3 |

ההשוואה הראשונה תהיה בין הערך בתא 0 לערך בתא 1, וערך המשתנה ששומר את המקום המקסימלי יישאר 0.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0** | 1 | **2** | 3 | 4 |
| 6 | 2 | 5 | 9 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0** | 1 | 2 | **3** | 4 |
| 6 | 2 | 5 | 9 | 3 |

ההשוואה השנייה תהיה בין הערך בתא 0 לערך בתא 2, וערך המשתנה ששומר את המקום המקסימלי יישאר עדיין 0.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | **3** | **4** |
| 6 | 2 | 5 | 9 | 3 |

ההשוואה השלישית תהיה בין הערך בתא 0 לערך בתא 3, וערך המשתנה ששומר את המקום המקסימלי ישתנה ל-3.

ההשוואה הרביעית תהיה בין הערך בתא 3 לערך בתא 4, וערך המשתנה ששומר את המקום המקסימלי יישאר 3.

רק בשלב הזה, לאחר שסרקנו את כל המערך פעם אחת, המשתנה maxPlaceישמור את המיקום של הערך הגדול ביותר במערך. זה הזמן להחליף את הערך שבמקום הנבחר עם הערך במקום האחרון.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 6 | 2 | 5 | 3 | 9 |

לאחר ההחלפה המערך ייראה כך:

כלומר בדיקה אחת של כל המערך מבטיחה שהמספר הגדול ביותר הגיע למקומו.

בשלב הזה נתחיל שוב: נאפס את maxPlace ונשווה שוב את הערך בתא 0 לערך בתא 1.

**בשונה** מהמיון הקודם, שבו סריקת התאים שכבר נמצאים במקומם לא פגעה בתהליך, במיון זה לא ניתן לסרוק את כל תאי המערך שוב – שכן המשתנה maxPlace יכיל תמיד בסיום הסריקה את מיקומו של הערך הגדול ביותר שמצאנו בפעם הקודמת. לכן באלגוריתם זה יש להתקדם בכל פעם תא אחד פחות מהפעם הקודמת.

אם נבדוק את כל המערך מספר פעמים כמספר התאים, בסוף התהליך המערך יהיה ממיון.

במקרה שלנו אלו יהיו ההעברות:

**6** **2** 5 9 3 -> **6** 2 **5** 9 3 -> **6** 5 2 **9** 3 -> 6 5 2 **9** **3** - swap - > 6 5 23 **9**

**6** **5** 2 3 9 -> **6** 5 **2** 3 9 -> **6** 5 2 **3** 9 - now swap - > 3 5 2 **6** **9**

**3** **5** 2 6 9 -> 3 **5** **2** 6 9 - swap - > 3 2 **5** **6** **9**

**3** **2** 5 6 9 - swap - > 2 **3** **5** **6** **9**

- > **2** **3** **5** **6** **9**

במשימה הבאה, נממש את המיון בסביבת העבודה ונציג אותו בשלבים.



### משימה 14: מיון בחירה

נכתוב פעולה שמקבלת מערך של מספרים ממשיים וממיינת אותו בעזרת מיון בחירה.

**פתרון**

האלגוריתם ידוע וכולל בדיקה והחלפה של הפעולות שכבר ביצענו. האתגר העיקרי יהיה להימנע מלבדוק את חלק המערך שכבר מוין. לאיזה ערך נכון להגביל בכל פעם את תחום הבדיקה של אינדקס הלולאה?

כל בדיקה של המערך תיראה כך:

int maxPlace = 0;

for (int place = 0; place <= lastPlaceToSort; place++) {

if (numbers[place] > numbers[maxPlace]) {

maxPlace = place;

}

}

C#

int maxPlace = 0;

for (int place = 0; place <= lastPlaceToSort; place++) {

if (numbers[place] > numbers[maxPlace]) {

maxPlace = place;

}

}

לאחר הבדיקה תתבצע ההחלפה עם המקום האחרון שנבדק, כלומר:

double temp = numbers [maxPlace];

numbers [maxPlace] = numbers [lastPlaceToSort];

numbers [lastPlaceToSort] = temp;

C#

double temp = numbers [maxPlace];

numbers [maxPlace] = numbers [lastPlaceToSort];

numbers [lastPlaceToSort] = temp;

כעת צריך להקטין את lastPlaceToSort ב-1, ולאחר מכן לחזור על כל התהליך שוב. כלומר נבצע לולאה שחוזרת על עצמה כאורך המערך, ובכל פעם המקטינה ב-1 את lastPlaceToSort ומאפסת שוב את maxPlace.

נכתוב את הפעולה כולה, כולל הלולאה שמקטינה את המקום האחרון לבדיקה:

public static void selectionSort(double[] numbers) {

for (int lastPlaceToSort = numbers.length - 1; lastPlaceToSort > 0; lastPlaceToSort--) {

int maxPlace = 0;

for (int place = 0; place <= lastPlaceToSort; place++) {

if (numbers[place] > numbers[maxPlace]) {

maxPlace = place;

}

}

double temp = numbers[maxPlace];

numbers[maxPlace] = numbers[lastPlaceToSort];

numbers[lastPlaceToSort] = temp;

}

}

C#

public static void SelectionSort(double[] numbers) {

for (int lastPlaceToSort = numbers.Length - 1; lastPlaceToSort > 0; lastPlaceToSort--) {

int maxPlace = 0;

for (int place = 0; place <= lastPlaceToSort; place++) {

if (numbers[place] > numbers[maxPlace]) {

maxPlace = place;

}

}

double temp = numbers[maxPlace];

numbers[maxPlace] = numbers[lastPlaceToSort];

numbers[lastPlaceToSort] = temp;

}

}

נבדוק את התוכנית שלנו על הערכים ההתחלתיים הללו:

10.0, 7.0, 1.0, 9.0, 6.0, 10.0, 8.0, 2.0, 8.0, 5.0, 5.0, 3.0

נקבל את המערך הממוין הבא:

1.0, 2.0, 3.0, 5.0, 5.0, 6.0, 7.0, 8.0, 8.0, 9.0, 10.0, 10.0

לצורכי בקרה אפשר להדפיס שלבי ביניים. לדוגמה נראה את המצב לאחר הבדיקות הראשונות. שימו לב שלפני כל בדיקה המערך התעדכן:

10.0 7.0 1.0 9.0 6.0 10.0 8.0 2.0 8.0 5.0 5.0 3.0

max found in place 0, with the amount of: 10.0

3.0 7.0 1.0 9.0 6.0 10.0 8.0 2.0 8.0 5.0 5.0 10.0

max found in place 5, with the amount of: 10.0

3.0 7.0 1.0 9.0 6.0 5.0 8.0 2.0 8.0 5.0 10.0 10.0

max found in place 3, with the amount of: 9.0

##### סיכום סוגי סריקה של מערך

לפני שנמשיך נסכם את שני סוגי סריקת המערך שכבר למדנו.

* הסוג הראשון הוא **סריקה של איברי המערך בעזרת לולאת for**. בלולאות מסוג זה הנחת העבודה היא סריקה של כל האיברים.
* הסוג השני הוא **סריקה של המערך עד קיום תנאי מבוקש בעזרת לולאת while**. לולאה זו שונה מהקודמת בכך שייתכן שלא כל איבריה ייבדקו.
* קיים סוג סריקה נוסף, והוא **סריקה שלא בעזרת אינדקס אלא בהתייחסות למערך כקבוצה**. במשימות הקרובות נתנסה בשלוש הסריקות הללו.



### משימה 15: מעקב אחר תוכנית נתונה

נתון המערך myArray שערכיו הם {12,0,-4,8,0,-2,0}; והמונה counter מאותחל ל-0.

מה יהיה הפלט לאחר השורות הבאות?

רמז: בדקו האם הלולאה השלימה סריקה של כל איברי המערך ושינתה אותם בהתאם לתנאי.

int counter = 0;

for (int place = 0; place < myArray.length; place++) {

if (myArray[place] == 0) {

counter++;

myArray[place] = 1;

}

}

System.out.println(counter);

for (int place = 0; place < myArray.length; place++) {

System.out.print(myArray[place] + " ");

}

System.out.println();

C#

int counter = 0;

for (int place = 0; place < myArray.Length; place++) {

if (myArray[place] == 0) {

counter++;

myArray[place] = 1;

}

}

Console.WriteLine(counter);

for (int place = 0; place < myArray.Length; place++) {

Console.Write(myArray[place] + " ");

}

Console.WriteLine();



### משימה 16: מעקב אחר תוכנית נתונה

נתון המערך myArray שערכיו הם {12,1,-4,8,1,-2,1};. מה יהיה הפלט לאחר השורות הבאות?

int placeInMyArray = 0;

while (placeInMyArray < myArray.length && myArray[placeInMyArray] > 0) {

myArray[placeInMyArray] = 0;

placeInMyArray++;

}

for (int place = 0; place < myArray.length; place++) {

System.out.print(myArray[place] + " ");

}

System.out.println();

C#

int placeInMyArray = 0;

while (placeInMyArray < myArray.Length && myArray[placeInMyArray] > 0) {

myArray[placeInMyArray] = 0;

placeInMyArray++;

}

for (int place = 0; place < myArray.Length; place++) {

Console.Write(myArray[place] + " ");

}

Console.WriteLine();



### משימה 17: מעקב אחר תוכנית נתונה

במשימה זו תתנסו בסריקה מהסוג השלישי, הסורקת את כל המערך ומאפשרת לבדוק את ערכיו אך לא לשנות אותם.

נתון המערך myArray שערכיו הם {0,0,-4,8,1,-2,1}; והמונה counter מאותחל ל-0.

מה יהיה הפלט לאחר השורות הבאות:

for (int value: myArray) {

if (value == 0) {

counter++;

value = 1;

}

}

System.out.println(counter);

for (int value: myArray) {

System.out.print(value + " ");

}

System.out.println();

C#

foreach (int value in myArray)

{

if (value == 0)

{

counter++;

}

}

הלולאה המוצגת כאן נקראת לולאת **for-each**,והיא מאפשרת לסרוק את המערך ולגשת למידע שבכל איבר בו, אבל משתנה הלולאה לא מצביע על הערכים עצמם אלא הם "מועתקים" למשתנה. לכן אי אפשר לשנות אותם.

### מערך דו-ממדי

ראינו בפרק זה שמערך מקל עלינו לשמור נתונים ולאחזר אותם בקלות באמצעות מציין מיקום (אינדקס) בתוך משתנה שאפשר לסרוק בלולאה. לעיתים השימוש במערך חד-ממדי טוב מהשימוש במשתנים בודדים, אבל גם אותו אפשר לשפר.

צפו בסרטון [*OCADO Warehouse Groceries Packing Robots*](https://www.youtube.com/watch?v=4DKrcpa8Z_E) ב-youtube והשיבו על השאלות:

1. כיצד לדעתכם ניתן לשמור את נתוני המוצרים בכל ארגז במחסן בכלים שלמדנו עד כה?
2. בדקה 01:35 מופיעה שקופית ובה האותיות X ו-Y. מה משמעותן לדעתכם?

בסרטון נראו רובוטים הנעים בארבעה כיוונים מעל משטח של ארגזים. לפני שנלמד כיצד מגדירים משטח כזה בג'אווה, ניזכר במשחק המוכר איקס-עיגול, או tic tac toe (למי שרוצה להתאמן, כתיבה של הרצף tic tac toe בגוגל תביא אתכם למשחק ולהסבר בוויקיפדיה). מדובר במשחק פשוט יחסית למימוש, ולמעשה הידע שלכם עד כה מספיק כדי לממש אותו.

את התאים במשחק ניתן לממש בעזרת תשעה משתנים, או בעזרת מערך עם תשעה תאים:

מימוש הלוח על ידי מערך חד-ממדי:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  | X |  | O |  | X |  |  |

מימוש הלוח על ידי 9 משתנים:

row1Colomn1=” “; row1Colomn2=” “; row1Colomn3=”X“;

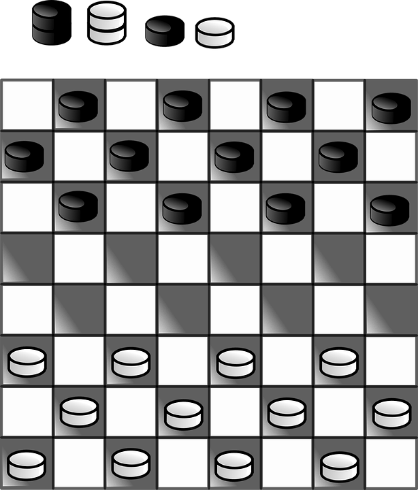
row2Colomn1=” “; row2Colomn2=”O“; row2Colomn3=” “;

row3Colomn1=”X“; row3Colomn2=” “; row3Colomn3=” “;

שתי הדרכים שונות זו מזו באופן שהן בודקות ניצחון, אבל בשתיהן אפשר לבדוק זאת. למשל, אפשר לבדוק ניצחון באלכסון המשני במימוש של משתנים רגילים באמצעות השוואה בין שלושה משתנים בדידים:

row1Colomn3, row2Colomn2, row3Colomn1.

ואילו במקרה של מערך, האלכסון הראשי מיוצג באמצעות המקומות 2, 4 ו-6, ולכן נוכל להשוות בין board[2],board[4],board[6].

אך במקרים רבים, בעיקר כאשר המבנה במציאות דומה לכך, יהיה נוח יהיה להגדיר מבנה חדש – המערך הדו-ממדי. **מערך דו-ממדי** ניתן לדימוי כטבלה.

ייתכן שאתם מכירים זאת ממשחק שחמט או דמקה. לכל משבצת יש שני מציינים: מציין השורה ומציין העמודה.

למשל, בלוח הדמקה הייצוג לכל משבצת נתון על ידי שני מציינים – מספר 1 עד 8 מייצג שורה, ואות אנגלית A עד H מייצגת עמודה.

### מערך דו-ממדי כמערך חד-ממדי של מערכים חד-ממדיים

כזכור, הגדרנו בתוכנת מרוץ הצבים מערך של עשרה עצמים מסוג צב:

Turtle[] turtles = new Turtle[10];

גם מערך הוא עצם, ולכן אם נרצה ליצור מערך של עשרה עצמים שלמים באורך 8, ככל הנראה נרשום כך:

int[][] numbers = new int[8][10];

ההגדרה הזו נראית מתאימה, אבל בפועל בג'אווה התוצאה שלה תהיה שונה מהמתוכנן. מסיבות שלא נדון בהן עכשיו, ההגדרה הזו לא תייצר עשרה עצמים מסוג מערך באורך 8, אלא דווקא שמונה עצמים מסוג מערך באורך 10.

ההגדרה הנכונה תהיה בכתיבה הבאה:

int[][] numbers = new int[10][8];

בהגדרת מערך דו-ממדי המספר הראשון משמאל יהיה מספר המערכים, והמספר השני יהיה מספר התאים בכל מערך.

אם נחזור לייצוג של לוח, אז המספר הראשון יהיה מספר השורות והשני מספר העמודות.

מספר השורות

מספר העמודות

ההגדרה הבאה:

 int[][] board = new int[6][7];

תיצור מערך דו-ממדי עם שש שורות ושבע עמודות שיתאים ללוח משחק של ארבע בשורה.

התאים במערכים בכל שורה יאותחלו אוטומטית בערך 0.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |

אפשר לדמות את המערך הדו-ממדי לטבלה שאפשר למלא במספרים דרך פנייה לתאים מסוימים. למשל ההוראה:

board [2][4] = 1;

תפנה לתא שבשורה 2 ובעמודה 4, ותשים בתוכו את הערך 1.

כך נוכל למלא את הלוח ולבדוק ניצחון.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 2 | 0 | 0 | 0 | 2 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 3 |
| 0 | 0 | 2 | 2 | 1 | 0 | 0 | 4 |
| 0 | 0 | 1 | 2 | 2 | 1 | 0 | 5 |

למשל בלוח הזה המנצח הוא שחקן מספר 1.

התאים שמגדירים ניצחון הם 4 תאים רצופים שבהם מופיע המספר 1. התאים הם: board [2][4],

board [3][3], board [4][2]

ו-board [5][1].

בפועל מה שקורה הוא שהעצם board מצביע על מערך של 6 תאים, שכל אחד מהם מצביע על מערך של 7 תאים של שלמים. כך זה נראה:

board

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |
| --- |
|  |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 1 | 2 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 2 | 2 | 1 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 1 | 2 | 2 | 1 | 0 |

אפשר לתפוס מערך דו-ממדי כאילו המערכים ממוקמים זה מתחת לזה, כי הם מייצגים עבורנו את שורות המשחק.

### בדיקות של מערך דו-ממדי

המערכים שנלמד עליהם כאן הם מערכים בעלי תצורה מלבנית.

במשימות הבאות נכיר מספר פעולות נפוצות על מערכים דו-ממדיים ונציג גם מספר מקרים פרטיים מעניינים.

היכנסו למשחק [tic-tac-toe](https://www.math10.com/en/math-games/tic-tac-toe/tic-tac-toe.html).   
(בכתובת <https://www.math10.com/en/math-games/tic-tac-toe/tic-tac-toe.html> )

שחקו בגרסת הלוח המוכרת של 3X3, ולאחר מכן נסו גם את 7X7.

מי שמעוניין לאתגר את עצמו במשימה קשה (אך אפשרית!), יכול לנסות לתכנן ולתכנת בעצמו אסטרטגיה למשחק כזה.

**תכננו** אלגוריתם שמקבל מערך המייצג מצב במשחק ואת השחקן בתור הקרוב, ומציע מהלכי משחק עבור השחקן כדי שינצח.



### משימה 18: הדפסה של ערכי מערך דו-ממדי כטבלה

חברת פיתוח משחקים החליטה להכין פעולות כלליות להדפסת לוחות משחק בגדלים שונים. עליכם לכתוב פעולה שמקבלת מערך דו-ממדי של מחרוזות המייצג לוח משחק מלבני.

בתאי המערך מחרוזות בגודל זהה. הפעולה תדפיס את ערכי המערך בהתאם לשורות ולעמודות בלוח.

ניתוח המשימה:

**אין קלט**: מקבלים מערך דו-ממדי נתון של מחרוזות בגודל זהה.

**פלט**: הדפסת המערך כלוח.

הפעולה תעבור על המערך בלולאה מקוננת (לולאה בתוך לולאה). הלולאה החיצונית תתקדם בכל פעם לשורה הבאה, והלולאה הפנימית תתקדם בכל פעם לעמודה הבאה (לתא הבא) בשורה.

הדפסת שורה היא פעולה פשוטה ומוכרת ממערך חד-ממדי. ההבדל הוא שהפעם המערך החד-ממדי הוא תא במערך הדו-ממדי. למשל המערך של השורה 0 הוא: board[0].

הדפסה של המערך תיראה כך:

for (int place = 0; place < board[0].length; place++) {

System.out.print(board[0][place] + " ");

}

System.out.println();

C#

for (int place = 0; place < board[0].Length; place++) {

Console.Write(board[0][place] + " ");

}

Console.WriteLine();

**שימו לב** שהדפסה זו זהה לחלוטין להדפסה של מערך חד-ממדי, רק שהמערך נקרא board[0].

בסוף הדפסת תאי המערך נרד לשורה הבאה.

בדומה לכך עלינו להדפיס את board[1], board[2] וכן הלאה.

נשתמש בלולאה נוספת ונחליף את מספר השורה בכל פעם. הפעם נשתמש ב-row כמציין השורה.

public static void printInt2Darray(int[][] board) {

for (int row = 0; row < board.length; row++) {

for (int place = 0; place < board[row].length; place++) {

System.out.print(board[row][place] + " ");

}

System.out.println();

}

}

C#

public static void PrintInt2Darray(int[,] board)

{

int RowCount = board.GetLength(0); // מספר שורות

int ColCount = board.GetLength(1); // מספר עמודות

int row;

int col;

for (row = 0; row < RowCount; row++)

{

for (col = 0; col < ColCount; col++)

Console.Write(String.Format("{0}\t", board[row, col]));

Console.WriteLine();

}

}

שימו לב שבמערכים שלנו board[row].length הוא האורך, שהוא מספר התאים שבכל שורה, ובהתאמה גם מספר העמודות. יכולנו לכתוב גם board[0].length, שכן מספר התאים בכל שורה קבוע. board.length הוא מספר השורות.



משימה 19: הדפסת מערך דו-ממדי בדגש על מיקום

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

חברת פיתוח משחקים החליטה להכין פעולות כלליות להדפסת לוחות משחק **בגדלים שונים**. עליכם לכתוב פעולה שמקבלת מערך דו-ממדי של שלמים המייצג לוח משחק מלבני אשר חלק ממשבצותיו לבנות וחלקן שחורות. הפעולה לא תדפיס את ערכי המערך, אלא לוח לבן שחור לסירוגין על פי ערכי התאים. התא board[0][0] תמיד ייצבע בלבן.

למשל לוח של 8X8 יודפס כך:

בגלל שבחלונית הקונסול לא ניתן להדפיס כמו במסך הגרפי, יש להדפיס רווח במקום צבע לבן ו-\* במקום צבע שחור.

**שימו לב!** חשוב לא להרוס את המערך שמתקבל.

פתרון

עקרון ההדפסה לא משתנה, אך נשים לב שכעת ההחלטה מה יודפס לא תלויה בערך התא אלא במציינים של השורה והעמודה. מחקר קטן יגלה שכאשר סכום המציינים זוגי המשבצת צריכה להיות שחורה, וכאשר סכומם אי-זוגי היא צריכה להיות לבנה.

ומכאן המימוש:

public static void printBW2Darray(int[][] board) {

for (int row = 0; row < board.length; row++) {

for (int col = 0; col < board[row].length; col++) {

if ((row + col) % 2 == 0)

System.out.print(1);

else

System.out.print(0);

}

System.out.println();

}

}

C#

public static void printBW2Darray(int[,] board)

{

for (int row = 0; row < board.GetLength(0); row++)

{

for (int col = 0; col < board.GetLength(1); col++)

{

if ((row + col) % 2 == 0)

Console.Write(1);

else

Console.Write(0);

}

Console.WriteLine();

}

}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

כפי שראינו במשימות האחרונות, המקרה הפרטי של מערך ריבועי שבו מספר השורות שווה למספר העמודות הוא נפוץ למדי. במערכים כאלו ניתן להגדיר אלכסונים.

האלכסון הראשי יורד מהפינה השמאלית העליונה לפינה הימנית התחתונה, והאלכסון המשני יורד מהפינה הימנית העליונה לפינה השמאלית התחתונה.

אפשר לראות הבדל בין מערך ריבועי בעל צלע אי-זוגית למערך בעל צלע זוגית.



משימה 20: סכום חלקי במערך דו-ממדי

במבחן כניסה לקורס "חשיבה יצירתית" באוניברסיטה התבקשתם לכתוב פעולה שמקבלת מערך ריבועי בגודל לא ידוע ומחזירה את סכום ערכי האלכסון הראשי שלו.

הדבר לא נאמר במפורש בשאלה, אבל חשבתם שבקורס על יצירתיות כדאי לכתוב פתרון ייחודי לכם, ואולי גם שיהיה יעיל במיוחד וישאיר חותם שונה משל האחרים.

ניתוח הבעיה:

הפתרון נראה פשוט: אפשר לסרוק את כל המערך ולהתמקד בערכים שצריך לסכום, כלומר באלכסון הראשי השורה (row) שווה לעמודה (col). כך זה נראה:

for (int row = 0; row < board.length; row++) {

for (int col = 0; col < board[row].length; col++) {

if (row == col)

sum += board[row][col];

}

}

C#

int sum = 0;

for (int row = 0; row < board.GetLength(0); row++)

{

for (int col = 0; col < board.GetLength(1); col++)

{

if (row == col)

sum += board[row,col];

}

}

אבל זו התשובה שכולם יגיעו אליה. איך נוכל לצמצם את הלולאות הללו? למעשה מספר הערכים באלכסון זהה למספר הערכים בשורה אחת, ובשיטה זו אנו סורקים את כל המערך שגודלו הוא גודל שורה בריבוע. אם נמצא שיטה לא לסרוק את כל המערך אלא רק את האלכסון, נחסוך בדיקות מיותרות רבות.

**פתרון:**

נסרוק רק את האלכסון! באלכסון מספר השורה ומספר העמודה שווים, והלולאה הבאה תעבור על האלכסון בלבד:

for (int rowCol = 0; rowCol < board.length; rowCol++)

sum += board[rowCol][rowCol];

C#

int sum = 0;

for (int rowCol = 0; rowCol < board.GetLength(1); rowCol++)

sum += board[rowCol,rowCol];

**רגע חושבים!**

מהשרטוט נראה שלפעמים יש הבדלים מבניים בין מערכים, והם משפיעים על השאלות שאנו יכולים לשאול. הבדל אחד לדוגמה הוא בין מערכים ריבועיים עם צלע זוגית לבין מערכים ריבועיים עם צלע אי-זוגית. במערכים אי-זוגיים, סכום איברי האלכסונים כולל חישוב נקודת האמצע פעמיים. האם נוכל להוריד פעם אחת?

למרות שאנו דנים במערך ריבועי, מעניין לבדוק מה קורה לאלכסון אם מספר השורות גדול ממספר העמודות, או מספר עמודות גדול ממספר השורות.



משימה 21: סכום חלקי במערך דו-ממדי

בשלב ב' של מבחן הכניסה לקורס "חשיבה יצירתית" באוניברסיטה התבקשתם לכתוב פעולה שמקבלת מערך ריבועי בגודל לא ידוע ומחזירה את סכום ערכי המסגרת שלו – כלומר סכום ערכי השורה הראשונה, ערכי השורה האחרונה, ערכי העמודה הימנית ביותר וערכי העמודה השמאלית ביותר.

גם בשאלה זו הודגש שסיכויי הקבלה לקורס יגדלו ככל שהפתרון יהיה יעיל יותר.

**פתרון**

ניתוח הבעיה:

הפתרון במקרה של בדיקת כל המערך נראה פשוט גם כאן – את הערכים שצריך לסכום נמצא בעזרת תנאים מתאימים, כלומר שורה 0 ושורה אחרונה, ועמודה 0 ועמודה אחרונה.

for (int row = 0; row < board.length; row++) {

for (int col = 0; col < board[row].length; col++) {

if (row == 0 || col == 0 || row == board.length - 1 || col == board[row].length - 1)

sum += board[row][col];

}

}

C#

int sum = 0;

int rowCount = board.GetLength(0); // מספר השורות

int colCount = board.GetLength(1); // מספר העמודות

int row, col;

for (row = 0; row <board.GetLength(0); row++)

{

for (col = 0; col<board.GetLength(1); col++)

{

if (row == 0 || col == 0 || row == board.GetLength(0) – 1  
 || col == board.GetLength(1) - 1)

{

sum += board[row, col];

}

}

אבל שוב זו תשובה שכולם יגיעו אליה. איך ניתן לצמצם את הלולאות הללו?

מספיק שנסרוק שתי שורות ושתי עמודות. אבל יש להיזהר עם הפינות, שלא נסכום אותן פעמים.

**פתרון:**

נסרוק את השורה הראשונה והאחרונה באותה לולאה:

for (int col = 0; col < board[0].length; col++)

sum = sum + board[0][col] + board[board.length - 1][col];

for (int row = 1; row < board.length - 1; row++)

sum = sum + board[row][0] + board[row][board.length - 1];

C#

int sum = 0;

int row, col;

for (col = 0; col < board.GetLength(0); col++)

sum = sum + board[0, col] + board[board.GetLength(0)- 1, col];

for (row = 1; row < board.GetLength(1) - 1; row++)

sum = sum + board[row,0] + board[row, board.GetLength(1) - 1];

###### נסכם: אם נסמן את מספר התאים בשורה ב-n, במקום לסרוק n2 תאים בעזרת n2 צעדי לולאה ולבצע n2 בדיקות ו-4n+4 פעולות סכימה, נסרוק כך רק 4n-4 תאים ונבצע 4n+4 פעולות סכימה בעזרת 2n+2 צעדי לולאה, ולא נבצע בדיקות כלל.

כלומר, אם מדובר במערך של 100X100, במקום 20,404 פעולות נבצע בדרך זו רק606 פעולות. נרחיב עוד בנושא יעילות בפרק שעוסק בכך בהמשך.

### מערך דו-ממדי של עצמים

ראינו כבר שעבודה עם מערכי עצמים אינה שונה מהותית מעבודה עם מערכים רגילים. ההבדל העיקרי הוא שבבניית המערך ובשליפת נתונים מהמערך נדרשת פנייה לפעולה פנימית של העצם.



משימה 22: השוואה בין ערכים סמוכים במערך

הצטרפתם למנהל מקרקעי ישראל וקיבלתם קצת חומר רקע על חלוקת הקרקעות בישראל. מסתבר שהחלוקה מתבססת על שיטת טורנס (Torrens) שבה השתמשו באימפריה הבריטית. לפי שיטה זו, כל הקרקע בישראל מחולקת בחלוקה ראשית לגושים ובחלוקה משנית לחלקות. יש חלקות שהן בבעלות המדינה, אחרות בבעלות פרטית ויש גם חלקות בבעלות של חברות.

גוש מוגדר כעצם בתוכנות הניהול של מנהל מקרקעי ישראל.

נתונה המחלקה Block.

Block

serialNumber – ערך מטיפוס שלם המייצג את מספר הגוש במנהל מקרקעי ישראל

numberOfOwners – ערך מטיפוס שלם המייצג את מספר בעלי החלקות בגוש

maxHight – ערך מטיפוס שלם המייצג את גובה המיקום הגבוה ביותר בגוש

lastTransaction – ערך מטיפוס תאריך, המייצג את תאריך העסקה האחרונה שנעשתה בגוש

אנו לא נזדקק לכל התכונות שלה אלא רק לאחת מהן – maxHight. אנחנו מחפשים במפה את המקומות הגבוהים ביותר ביחס לשכניהם, והם יוגדרו כמקסימום מקומי במטרה להעמיד עליהם אנטנות סלולריות.

לפניכם המפה של אחד מהאזורים הללו ובה הגבהים המקסימליים – שהם התאים שהערך בהם גבוה מהערכים בכל שמונת התאים הסמוכים להם. הגבהים המקסימליים מסומנים במפה (תאים על הגבול לא יכולים להיות מקסימום מקומי).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
| 121 | 122 | 5 | 121 | 122 | 5 | 122 | 5 | 121 | 122 | 5 | 121 | 122 | 5 | 122 | 5 | 23 | 0 |
| 112 | 142 | 55 | 112 | 142 | 55 | 366 | 12 | 36 | 232 | 55 | 112 | 142 | 55 | 142 | 55 | 452 | 1 |
| 243 | 321 | 56 | 243 | 321 | 56 | 24 | 134 | 24 | 121 | 56 | 243 | 31 | 56 | 321 | 56 | 56 | 2 |
| 212 | 231 | 332 | 212 | 231 | 332 | 22 | 45 | 22 | 312 | 302 | 212 | 231 | 312 | 231 | 332 | 35 | 3 |
| 211 | 231 | 13 | 211 | 231 | 13 | 245 | 245 | 245 | 321 | 13 | 211 | 231 | 13 | 231 | 13 | 356 | 4 |
| 221 | 123 | 145 | 221 | 123 | 145 | 256 | 32 | 256 | 111 | 145 | 221 | 123 | 145 | 123 | 145 | 25 | 5 |
| 232 | 12 | 366 | 232 | 12 | 366 | 12 | 321 | 232 | 12 | 36 | 272 | 12 | 66 | 12 | 36 | 35 | 6 |
| 121 | 134 | 24 | 121 | 134 | 24 | 134 | 24 | 121 | 134 | 24 | 112 | 142 | 55 | 134 | 24 | 17 | 7 |
| 312 | 45 | 22 | 312 | 45 | 22 | 45 | 22 | 312 | 45 | 22 | 243 | 333 | 56 | 45 | 22 | 86 | 8 |
| 321 | 245 | 245 | 321 | 245 | 245 | 245 | 245 | 371 | 245 | 245 | 212 | 231 | 332 | 245 | 245 | 89 | 9 |
| 111 | 32 | 256 | 111 | 32 | 256 | 32 | 26 | 111 | 32 | 256 | 211 | 231 | 13 | 32 | 256 | 58 | 10 |
| 76 | 234 | 224 | 76 | 234 | 224 | 234 | 224 | 76 | 234 | 224 | 221 | 123 | 145 | 234 | 224 | 58 | 11 |
| 65 | 53 | 267 | 65 | 53 | 267 | 53 | 267 | 65 | 53 | 267 | 112 | 142 | 55 | 53 | 267 | 90 | 12 |
| 50 | 245 | 325 | 50 | 245 | 325 | 245 | 340 | 50 | 245 | 325 | 50 | 245 | 325 | 245 | 325 | 56 | 13 |
| 44 | 35 | 25 | 44 | 35 | 25 | 35 | 25 | 44 | 35 | 25 | 44 | 35 | 25 | 35 | 25 | 58 | 14 |

**כתבו** תוכנית המקבלת מערך דו-ממדי של גושים (עצמים מסוג Bloc) ומדפיסה את המיקומים של כל הגושים שהגובה המקסימלי בהם גדול יותר מערכי כל הגושים שסביבם. גבהים מקסימליים כאלו נקראים מקסימום מקומי. אומנם למערך כולו יש רק ערך מקסימלי אחד, אבל גובה מקסימלי מוגדר באופן יחסי לסביבתו – כלומר מתוך שמונה התאים שמקיפים אותו, הוא הגבוה ביותר.

למפה שבדוגמה יודפסו הצמדים: (1,7)(1,10)(3,11)(3,14)(4,7)(6,5)(6,9)(6,11)(6,14)(7,2)(8,4)(9,8)(9,13)(10,1)(10,11)(13,1)(13,3)(13,6)(13,9)(13,11)(13,14)

ניתוח הבעיה:

יש להימנע מבדיקת המסגרת, כלומר לעבוד מ-1 עד 2 פחות מאורך המערך.

יש לבדוק לכל תא את הסובבים אותו – כלומר שלושה תאים בשורה מעליו, שלושה בשורה מתחתיו, אחד משמאלו ואחד מימינו. אנו משתמשים למעשה בערך אחד של עצם מסוג Block. נצא לדרך:

public static void printAllLocalMax(Bloc[][] map) {

for (int row = 1; row < map.length - 1; row++) {

for (int col = 1; col < map[row].length - 1; col++) {

int value = map[row][col].getMaxHeight();

if (value > map[row - 1][col - 1].getMaxHeight() &&

value > map[row - 1][col].getMaxHeight() &&

value > map[row - 1][col + 1].getMaxHeight() &&

value > map[row + 1][col - 1].getMaxHeight() &&

value > map[row + 1][col].getMaxHeight() &&

value > map[row + 1][col + 1].getMaxHeight() &&

value > map[row][col - 1].getMaxHeight() &&

value > map[row][col + 1].getMaxHeight())

System.out.println(“(“+row + ”, ”+col + ”)”);

}

}

}

C#

public static void PrintAllLocalMax(Bloc[][] map) { //

for (int row = 1; row < map. board.GetLength(1) 1; row++) {

for (int col = 1; col < map[row]. board.GetLength(1); col++) {

int value = map[row,col].GetMaxHeight();

if (value > map[row - 1][col - 1].GetMaxHeight() &&

value > map[row – 1,col].GetMaxHeight() &&

value > map[row – 1,col + 1].GetMaxHeight() &&

value > map[row + 1,col - 1].GetMaxHeight() &&

value > map[row + 1,col].GetMaxHeight() &&

value > map[row + 1,col + 1].GetMaxHeight() &&

value > map[row,col - 1].GetMaxHeight() &&

value > map[row,col + 1].GetMaxHeight())

Console.WriteLine("("+row + " , "+col + ")");

}

}

}

שאלות למחשבה:

1. מה היה קורה אם לא היינו תוחמים נכון את גבולות הבדיקה?
2. מה המשמעות של חריגה מגבולות המערך?
3. אילו שיטות נוספות תוכלו להציע להשוואה זו?

##### סיכום פרק מערכים

* **מערך** הוא עצם הבנוי מקבוצת תאים מטיפוס מסוים. הטיפוס ומספר התאים נקבעים בהגדרת המערך.
* אפשר **להגדיר מערך** בשלוש צורות:

1. קביעת הטיפוס והשם בהוראה אחת, וקביעת הגודל בהוראה נוספת אחריה.

int[] name;

name = new int[9];

1. קביעת הטיפוס, השם והגודל בהוראה אחת.

int[] name = new int[9];

1. קביעת הטיפוס והשם ואתחול המערך בהוראה אחת

int [] name = {1,2,3,4,5};

* **אינדקס (מציין)** הוא מספר המייצג מיקום במערך.
* **תא במערך** הוא משתנה שאפשר לגשת אליו לפי מיקומו במערך. המיקומים מתחילים תמיד במיקום 0.

הערך של התא name[3], המערך שהוגדר בסעיף הקודם הוא 4.

* **העברת מערך לפעולה** היא העברת מצביע על המערך המקורי בפעולה הראשית. המשמעות היא שכל שינוי בערכים במהלך הפעולה ייראה בפעולה הקוראת כשהפעולה תסתיים.
* **for-each** היא שיטת בדיקה של תאי מערך (תחביר השיטה מוגדר רק מגרסה 5 של ג'אווה ומעלה).
* **חיפוש סדרתי** הוא בדיקת ערכי המערך בהתקדמות של תא אחר תא.
* **חיפוש בינארי** הוא בדיקת ערכי מערך ממוין תוך הקטנת אזור החיפוש במחצית בכל מהלך.
* **מיון מערך** הוא סידור ערכי המערך על פי מפתח מסוים, למשל מהערך הגדול לקטן, לפי סדר אלפביתי, לפי תאריך ועוד.
* **יעילות** היא מידת צריכת המשאבים להשגת מטרה מסוימת, כמו מספר פקודות או גודל זיכרון.
* **מערך דו-ממדי** הוא מערך של מערכים:

מספר השורות

מספר העמודות

int[][] board = new int[6][7];

* **מערך ריבועי** הוא מקרה פרטי של **מערך מלבני**, ובו מספר השורות שווה למספר העמודות. במערך זה ניתן להגדיר אלכסון ראשי ומשני.
* **אלכסון ראשי** מכיל את כל התאים מסוג table[i][i]
* **אלכסון משני** מכיל את כל התאים מסוג table[i][table.lnength - 1 - i]

אם גודל צלע המערך אי-זוגי, יש תא אחד הנמצא בשני האלכסונים והוא אמצע הריבוע.

### חמישה בחמש

1. הצפנת מסכה – שימוש במערך בינארי כמפתח להעברת מסרים.
2. קצת מעבר – מדוע מיקום איבר במערך מתחיל במספר 0?
3. מידע באפס ידע – הוכחת פתרון מבלי לרמוז מהו.
4. מטריצה מתמטית – מטריצה כמקדמי מערכת משוואות.
5. איך עבד, עובד פקס – bitmap.
6. משחק החיים – סימולציה.
7. הסרט מטריקס – מקור השם ומשמעותו בסרט.
8. טלפון סלולרי – מה מקור השם סלולרי.
9. שיטות ריצוף – מערך דו-ממדי כאוסף אריחים בני יותר מתא אחד.
10. הנפה של ארטוסטנס – מגבלות המערך.
11. ריבוע קסם – בדיקה ויצירה.

# פרק 6: תכנות מונחה עצמים



משימה 1: בואו נשחק

לרשותכם מחלקה בשם Player – שחקן .עבור השחקן הוגדרו פעולות הממשק הבאות:

|  |  |
| --- | --- |
| בנאי – פעולה בונה של המחלקה שחקן. הפעולה מקבלת פרמטר יחיד – שם השחקן, ומייצרת מופע של שחקן.  מספר הנקודות ההתחלתי של השחקן הוא 0. | Player(String name) |
| שאילתה – פעולה המחזירה את שם השחקן. | String getName() |
| שאילתה – פעולה המחזירה את מספר הנקודות שצבר השחקן. | int getPoints() |
| פקודה המשנה את מספר הנקודות שצבר השחקן (מוסיפה את points לניקוד שצבר השחקן עד כה). | void setPoints(int points) |
| שאילתה – פעולה המחזירה מחרוזת המתארת את השחקן. | String toString() |

במשימה זו עלינו לפתח אלגוריתם המדמה את המשחק הבא:

שני שחקנים מטילים קובייה, כל אחד בתורו.

כל שחקן מקבל את הניקוד לפי תוצאת הטלת הקובייה שלו.

המשחק מסתיים לאחר 16 סיבובים, ובסיום מכריזים מי המנצח.

**בצעו** את המשימה בשני שלבים:

1. **פתחו** אלגוריתם לפתרון הבעיה.
2. **ישמו** את האלגוריתם בתוכנית מחשב.

המשחק מסתיים לאחר 16 סיבובים, ובסיום מכריזים מי המנצח.

**בצעו** את המשימה בשני שלבים:

1. **פתחו** אלגוריתם לפתרון הבעיה.
2. **ישמו** את האלגוריתם בתוכנית מחשב.

לפניכם הצעה לפתרון המשימה והמימוש שלה.

**השוו** את הפתרון שלכם לפתרון זה.

public static void main(String[] args) {

Player p1 = new Player("yossi");

Player p2 = new Player("dafna");

Random rnd = new Random();

int res1, res2;

for (int k = 1; k <= 16; k++) {

res1 = rnd.nextInt(6) + 1;

p1.setPoints(res1);

res2 = rnd.nextInt(6) + 1;

p2.setPoints(res2);

}

if (p1.getPoints() > p2.getPoints())

System.out.println("The winner is " + p1.getName() + " with " + p1.getPoints());

else

if (p2.getPoints() > p1.getPoints())

System.out.println("The winner is " + p2.getName() + " with " + p2.getPoints());

else

System.out.println("It is a draw with " + p1.getPoints() + " " + p1.getName() + " " + p2.getName());

}

C#

public static void main(string[] args) {

Player p1 = new Player("yossi");

Player p2 = new Player("dafna");

Random rnd = new Random();

int res1, res2;

for (int k = 1; k <= 16; k++)

{

res1 = rnd.Next(6) + 1;

p1.SetPoints(res1);

res2 = rnd.Next(6) + 1;

p2.SetPoints(res2);

}

if (p1.GetPoints() > p2.GetPoints())

Console.WriteLine("The winner is " + p1.GetName() + " with " +

p1.GetPoints());

else

if (p2.GetPoints() > p1.GetPoints())

Console.WriteLine("The winner is " + p2.GetName() + " with " +

p2.GetPoints());

else

Console.WriteLine("It is a draw with " + p1.GetPoints() + " " +   
 p1.GetName() + " " + p2.GetName());

}

**היכנסו** לאינטראקציה הבאה (תסריט מצורף): מה מסתתר בתוך המחלקהPlayer ?

### סיכום ביניים:

עד היום הכרנו טיפוסי ערכים פשוטים המוגדרים בשפה כמו int ו-double, יצרנו משתנים אשר באפשרותם להכיל ערכים מטיפוסים אלה והשתמשנו בהם לפתרון בעיות. על ערכים פשוטים אלה אפשר היה לבצע פעולות שונות כמו חיבור, חיסור, מציאת יחס הסדר בין הערכים ועוד. הכרנו גם טיפוסי ערכים מורכבים לא-פשוטים הכלולים בשפה.

כמו כן הכרנו את הטיפוסים System, Scanner, Random, Math, שכונו בשם מחלקות (classes) ואת הפעולות הייחודיות להם. בנוסף, הכרנו גם את הטיפוס מחרוזת (String) המאפשר לטפל באוסף של תווים המאוגדים יחד. טיפוס נוסף מסוג זה הוא המערך (array), המאפשר לטפל באוסף של נתונים, לברר את אורכו, ולגשת ישירות לכל אחד מאיברי האוסף כדי לשלוף אותו, לעדכן אותו ועוד.

השתמשנו גם במחלקות מטיפוסים מורכבים שהוכנו מראש –Turtle, Bucket, Organ, ובתחילת הפרק גם המחלקה Player. גילינו כיצד המחלקה בנויה מבפנים וכיצד היא ממומשת.

### א. טיפוסים חדשים

כדי לפתח תוכנות מעניינות ומורכבות, לדוגמה משחקי מחשב, נרצה להגדיר בעצמנו טיפוסי עצמים חדשים נוספים. עבור כל טיפוס כזה יש להגדיר הן קבוצת ערכים מסוג מסוים והן אוסף של פעולות שניתן לבצע על ערכים אלה. דוגמאות לטיפוסים כאלה הן שחקנים במשחק, קלפים, קוביות משחק, גופים הנדסיים שונים, תאריכים ועוד. הטיפוסים החדשים מאופיינים על ידי הערכים שלהם והפעולות שניתן לבצע עליהם.

##### משימה 1: האורגן כמחלקה

**היזכרו** באורגן שפגשנו בפרקים 1 ו-4. ניתן לבצע פעולה של הקלטת רצף תווים, השמעת הרצף ולבסוף מחיקתו.

**השלימו** שלושה תווים לבחירתכם והאזינו לתוצאה:

Organ myOrgan = new Organ();

myOrgan.add(\_\_);

myOrgan.add(\_\_);

myOrgan.add(\_\_);

myOrgan.play();

myOrgan.Clear();

ב-JAVA, טיפוס חדש מסוג זה מוגדר על ידי **מחלקה** (**class**). כפי שמשתנים מטיפוסים פשוטים הכילו ערכים מסוג הטיפוס, כך גם משתנים מטיפוס מחלקה כלשהי יכילו ערכים מטיפוס המחלקה. משתנה מטיפוס Organ יכיל ערכים שונים מטיפוס Scanner מכיוון שכל אחד מהם שייך לטיפוס אחר, למחלקה אחרת.

משתנים מטיפוס מחלקה נקראים עצמים (objects).

האורגן myOrgan שבעזרתו ניגנתם כעת הוא עצם מטיפוס המחלקה Organ.

כדוגמה נוספת נתעמק בקלף שבו השתמשתם כששיחקתם במשחק הזיכרון בפרק 1. גם הקלף הוא טיפוס חדש המוגדר על ידי מחלקה, ותוכלו להשתמש במחלקה זו ליצירת מגוון משחקי מחשב הכוללים קלפים. בפרק זה נשכלל את הקלף ונרחיב את אפשרויות השימוש בו. לצורך כך נכתבה המחלקה Card, וכעת תוכלו ליצור עצמים חדשים מטיפוס זה.

|  |  |
| --- | --- |
| בנאי (פעולה בונה) של המחלקה קלף. הפעולה מקבלת פרמטר יחיד – צבע הקלף, ומאתחלת את המופע של הקלף החדש.  הפעולה מגרילה את מספר הקלף בין 1 ל-13.  הקלף נוצר במצב סגור.  צבע הקלף red / black. | Card(String color) |
| שאילתה המחזירה את מספר הקלף. | int getNum() |
| שאילתה המחזירה את צבע הקלף. | String getColor() |
| שאילתה המחזירה אמת אם הקלף פתוח, אחרת מחזירה שקר. | boolean getStatus() |
| פקודה המשנה את מצב הקלף בהתאם לערך המתקבל. | void setStatus(boolean status) |
| שאילתה המחזירה אמת אם מספר הקלף הנוכחי גדול ממספר קלף המתקבל כפרמטר. אחרת מחזירה שקר. | boolean isGreater(Card other) |
| שאילתה הפעולה מחזירה מחרוזת המתארת את הקלף. | String toString() |

לדוגמה, הפקודה:

Card c1;

מגדירה משתנה מטיפוס Card שיוכל להכיל עצם מטיפוס זה.

הפקודה:

c1 = new Card("red");

יוצרת עצם חדש מטיפוס זה שצבעו אדום ומספרו בין 1 ל-13. המספר נקבע באופן אקראי. העצם החדש נכנס למשתנה c1.



##### משימה 2: שמות קלפים

**צרו** שני קלפים נוספים ובחרו את צבעם (אדום או שחור black/red):

Card c2 = new Card(“\_\_\_\_\_\_\_\_\_\_\_\_”);

Card c3 = new Card(“\_\_\_\_\_\_\_\_\_\_\_\_”);

### ב. מצב של עצם

כיצד ניגשים לבנות מחלקה? במסגרת הגדרת המחלקה צריך להגדיר את מצב העצם.

לכל **עצם** (**object**) נקבע **מצב** (**state**) משלו בזמן יצירתו. הוראה ליצירת עצם מייצרת עצם חדש על פי המצב המוגדר בה.

את מצבו של עצם מטיפוס "קלף" נוכל לאפיין באמצעות מספר, צבע, וקביעה אם הוא פתוח או סגור. מצב הקלף נקבע בזמן יצירתו. במחלקה קלף מספרו של קלף חדש מוגרל בעת יצירתו לאחד הערכים שבין 1 ל-13, צבעו נקבע על ידי המשתמש, ומצבו ההתחלתי תמיד סגור כי כך הוגדר במשימה (יכולנו גם להחליט כי קלף חדש יהיה במצב פתוח). בדוגמה לעיל, בזמן יצירת הקלף c1 הוגרל המספר 5 (אפשר לגלות זאת רק באמצעות שאילתה מתאימה), צבעו נקבע ל- red, והקלף סגור.

עצמים שונים מאותו טיפוס יכולים כמובן להיות בעלי מצבים שונים: למשל לקלפים שונים יש מספרים שונים, צבעים שונים ומנחים שונים (פתוחים/סגורים). אך ייתכן כמובן שלעצמים שונים שנוצרו מאותה מחלקה יהיו מצבים זהים.



##### משימה 4: דומים ושונים

לפניכם שלושה קלפים, c1, c2, c3.

Card c1 = new Card(“black");

Card c2 = new Card(“red”);

Card c3 = new Card(“black”);

הנחה: מספרי שלושת הקלפים הם 9.

איזה זוג קלפים בעלי מצב שונה? (c1, c2)

איזה זוג קלפים בעלי מצב זהה? (c1, c3)

כפי שראינו בעבר, מצבם של העצמים יכול להשתנות במהלך תוכנית. כל עצם יכול לבצע פעולות המדווחות על מצבו או משנות אותו. לדוגמה, קלף שהיה סגור יכול להיפתח. הוראה שהופכת למשל קלף סגור לפתוח נקראת "פעולה".

ה**פעולות** (**methods**) שעצם מבצע יודעות מה מצבו ויכולות לאחזר ערכים המתארים את אותו או משנים אותו.

**רגע חושבים!**

אילו פעולות נוכל לבצע על מחלקה של קלף?

למשל, נוכל להגדיר עבור הטיפוס "קלף" את הפעולות הבאות: שאילתה לאחזור צבע הקלף, שאילתה לאחזור מספר הקלף, שינוי מצב הקלף מסגור לפתוח, השוואה בין המספרים של שני קלפים. כאשר עצם מטיפוס "קלף" מבצע, למשל, את פעולת אחזור הצבע של הקלף, מצבו של העצם מוכר לפעולה, ולכן היא יכולה למצוא את ערך הצבע ולהחזירו. זאת ללא קשר האם הקלף פתוח או סגור.



##### משימה 5: פעולות על קלפים

לרשותכם עומדת מחלקת קלף וכן מספר קלפים:

Card c1 = new Card(“red”);

c1.setStatus(true);

Card c2 = new Card(“black”);

…

עליכם לבצע עליה מספר פעולות:

1. **צרו** קלף חדש
2. **בחרו** את אחד הקלפים ו**פתחו** אותו (העבירו ממצב סגור לפתוח)
3. **אחזרו** את הצבע של שניים מהקלפים
4. **אחזרו** את מספרו של אחד הקלפים
5. **בדקו** מה מצב אחד הקלפים בסיום – האם הוא סגור או פתוח

**שימו לב!** העצם יבצע פעולה רק כאשר הוא מקבל הוראה מתאימה.

למשל, אם הגדרנו קלף חדש עם צבע “red”, כדי לבדוק מהו צבעו של הקלף עלינו להשתמש בשאילתה, בפעולת אחזור, ולא נוכל להסתמך רק על ההגדרה של הקלף.

בשלב ראשוני זה של עיסוק בעצמים, הדבר נראה מובן מאליו. אולם ככל שנתקדם ותבינו יותר לעומק את המאפיינים של עצמים ומחלקות, תתוודעו לחשיבותה של הערה זו.

כפי שראינו במשימה 1, מחלקה כוללת הגדרה של ייצוג מצבי העצמים, הפעולות במחלקה ואופן כתיבתן, ומנגנון המאפשר לייצר עצמים במצב התחלתי רצוי.

### ג. תוכנית

באופן כללי, תוכנית מכילה הגדרות של מחלקה אחת או מספר מחלקות. במהלך ביצועה נוצרים עצמים מטיפוס המחלקות. עצמים אלה מבצעים פעולות כאשר הם מקבלים הוראות. בנוסף, נכתבה מחלקה שבה מופיעה הפעולה הראשית (הקרויה main), שבה מתחיל ביצוע התוכנית.

בפעולה הראשית נוצרים עצמים מטיפוסי המחלקות, ואחר כך נשלחות לעצמים אלה הוראות לביצוע פעולות שונות. העצמים יכולים להעביר הודעות ביניהם. ביצוע הפעולות הנדרשות על ידי ההודעות מביא להשלמת המשימה של התוכנית.

Card c1 = new Card("red");

Card c2 = new Card("black");

if (c1.getNum() > c2.getNum())

System.out.println("The number of c1 is greater then c2");

else

if (c1.getNum() < c2.getNum())

System.out.println("The number of c2 is greater then c1");

else

System.out.println("The number do c1 equals to c2");

C#

Card c1 = new Card("red");

Card c2 = new Card("black");

if (c1.GetNum() > c2.GetNum())

Console.WriteLine("The number of c1 is greater then c2");

else

if (c1.GetNum() < c2.GetNum())

Console.WriteLine("The number of c2 is greater then c1");

else

Console.WriteLine("The number do c1 equals to c2");

באיור זה Test היא המחלקה הראשית. היא מכילה את קטע הקוד היוצר עצמים ממחלקת הקלף ומעבירה אל העצמים הודעות לביצוע פעולות שונות.

תכנות מונחה עצמים מתבצע בשני שלבים:

1. כתיבת מחלקות המגדירות טיפוסי עצמים חדשים, במקרה זה כתיבת המחלקה Card.
2. כתיבת פעולה ראשית היוצרת מהמחלקות עצמים ומפעילה אותם, במקרה זה הפעולה main הנמצאת במחלקה Test.

### ד. ממשק

לכל מחלקה מצורף מסמך המפרט כיצד להשתמש בה שנקרא ממשק (interface). בממשק מפורטות הפעולות שעצם יכול לבצע וכן בנאי (constructor) או מספר. פעולות אלה מגדירות דרכים לבניית עצמים. עבור כל פעולה מופיעה הכותרת (header) שלה, ובה שם הפעולה, טיפוסי הפרמטרים שהיא מקבלת וטיפוס הערך שהיא מחזירה. הממשק מכיל גם תיאור מילולי של הפעולה. בדרך כלל מכיל הממשק תיאור של בנאי אחד או יותר. ההסכם המחייב בשפת JAVA הוא ששם הבנאי זהה תמיד לשם המחלקה ואין פירוט של טיפוס הערך המוחזר ממנה. אין צורך לדעת כיצד מחלקה ממומשת כדי להשתמש בה, מספיק לדעת באיזה אופן משתמשים בה – כלומר די שנדע כיצד לייצר ממנה עצמים ולהפעיל אותם. עקרון זה נקרא הפרדה בין ממשק למימוש. כפי שנראה בהמשך, קיים מנגנון פשוט המייצר תיאור של ממשק המחלקה מהתיעוד הפנימי שלה.

**רגע חושבים!**

מה היתרון בעיקרון ההפרדה בין ממשק (interface) למימוש (implementation) בעבודה משותפת של מתכנתים רבים על אותה תוכנית?

המושגים ממשק, מימוש, תיעוד והפרדה בין ממשק למימוש אינם ייחודיים לתכנות אלא משמשים אותנו בטבעיות בתחומים רבים. לדוגמה, כאשר אנו קונים טלפון סלולרי חדש אין צורך שנדע מהו המבנה הפנימי שלו, מספיק שנקבל חוברת הדרכה עם פירוט כיצד להפעיל את כל האפשרויות שהמכשיר מציע.

בדומה לכך, כאשר אנו משתמשים בתוכנה מסוימת – למשל Word, אין צורך שנדע איך היא נכתבה וכיצד היא ממומשת, אלא מספיק לנו לדעת היכן להקליק עם העכבר ובאילו תפריטים לחפש את מה שדרוש לעבודתנו.

לפניכם הממשק של המחלקה "קלף" – Card. העמודה הימנית מציגה את תיאור הפעולה והעמודה השמאלית מציינת את כותרת הפעולה.

נדגיש: מכיוון שהפעולות מתבצעות על ידי העצם עצמו, אין צורך להזכיר את העצם בתיאורי הפעולות. העצם אינו פרמטר של פעולה אלא המפעיל שלה. יוצאת דופן היא הפעולה הבונה, שאינה פעולה של עצם כלשהו אלא מופעלת על ידי המחלקה, אך גם עבורה אין צורך בעצם כפרמטר.

בדקו עד כמה הבנתם את הנושא באמצעות השאלות הבאות:

עבור כל אחת מהטענות הבאות, קבעו האם היא נכונה או לא

שאלות נכון / לא נכון

ממשק המחלקה מציג את תכונות המחלקה ואת הפעולות המוגדרות עבור עצמים מטיפוס המחלקה – לא נכון

ממשק המחלקה מציג את תכונות המחלקה בלבד – לא נכון

ממשק המחלקה מציג את הפעולות שניתן לבצע על עצמים המוגדרים מטיפוס המחלקה - נכון

ממשק המחלקה Card

המחלקה Card מגדירה קלף לפי הערכים מספר, כותרת ומצב.

|  |  |
| --- | --- |
| בנאי (פעולה בונה) של המחלקה קלף. הפעולה מקבלת פרמטר יחיד – צבע הקלף, ומאתחלת את מצב הקלף החדש.  הפעולה מגרילה את מספר הקלף בין 1 ל- 13.  הקלף נוצר במצב סגור.  צבע הקלף red / black. | Card(String color) |
| שאילתה המחזירה את מספר הקלף. | int getNum() |
| שאילתה המחזירה את צבע הקלף. | String getColor() |
| שאילתה המחזירה אמת אם הקלף פתוח, אחרת מחזירה שקר. | boolean getStatus() |
| פקודה המשנה את מצב הקלף בהתאם לערך המתקבל. | void setStatus(boolean status) |
| שאילתה המחזירה אמת אם מספר הקלף הנוכחי גדול ממספר הקלף המתקבל כפרמטר, אחרת מחזירה שקר. | boolean isGreater(Card other) |
| שאילתה הפעולה מחזירה מחרוזת המתארת את הקלף. | String toString() |

מקובל לחלק את הפעולות המוגדרות במחלקה לשתי קטגוריות:

שאילתא – פעולה המחזירה ערך ללא שינוי מצב העצם

פקודה – פעולה המשנה את מצב העצם

פותרים ביחד – משימה 1:

עבור כל אחת מפעולות הממשק ציינו האם היא שאילתא או פקודה

### ה. הבנאי

הבנאי (constructor) בונה עצם חדש וקובע את מצבו. שמו של הבנאי הוא כשם המחלקה.

כך מתוארת פעולת הבנאי בממשק של המחלקה "קלף" – Card:

|  |  |
| --- | --- |
| בנאי של המחלקה קלף. הפעולה מקבלת פרמטר יחיד – כותרת הקלף ומאתחלת את מצב הקלף החדש.  הפעולה מגרילה את מספר הקלף בין 1 ל- 13  הקלף נוצר במצב סגור.  צבע הקלף “red” / “black” | Card (String color) |



משימה 2 – בדיקת טווח הקלפים:

השיבו על השאלה הבאה:

האם לאחר שימוש בבנאי עלינו לבדוק כי המספר שהתקבל עבור הקלף הוא אכן בטווח המוגדר בין 1 ל-13?

תשובה (מוסתרת): באחריות הבנאי לאתחל את העצם למצב התחלתי תקין. במקרה זה ההגרלה תוודא שמספר הקלף הוא בטווח המוגדר.

האם לאחר שימוש בבנאי צריך לבדוק כי צבע הקלף הוא אדום או שחור בלבד?

תשובה (מוסתרת): באחריות מי שמגדיר את העצם הנבנה לאתחל את העצם למצב התחלתי תקין.



משימה 3: יוצרים עצמים

### חלק א'

1. צרו עצם חדש מטיפוס המחלקה – קלף שצבעו red

פתרון :

new Card ("red");

הוראה זו יוצרת עצם חדש, כלומר מקצה מקום בזיכרון לעצם זה. מייד אחר כך מופעל הבנאי וקובע את מצבו של העצם בהתאם לערך הפרמטר. למשל, אם הוגרל המספר 8 – יווצר מופע של קלף שמספרו 8, צבעו אדום והוא סגור.

### חלק ב'

1. צרו עצם חדש מטיפוס קלף שצבעו הוא black.

פתרון :

new Card ("black");

הוראה זו יוצרת עצם חדש, כלומר מקצה עבורו מקום בזיכרון. מיד אחר כך מופעל הבנאי וקובע את מצבו של העצם בהתאם לערך הפרמטר.

לעצם שיצרנו ממחלקה מקובל לקרוא מופע (instance) של המחלקה. כל קלף מהווה מופע של המחלקה, ואת המחלקה מגדירים פעם אחת.

**רגע חושבים!**

כמה קלפים ניתן להגדיר מטיפוס המחלקה?

**תשובה:** יכולים להיות לה אינסוף מופעים – בכל פעם שמשתמשים בבנאי נוצר מופע חדש. בשלב זה של לימודיכם, יצירת עצמים נעשית בדרך כלל בתוך הפעולה הראשית האחראית על מבנה התוכנית. בהמשך

תלמדו כי מחלקה יכולה להכיל בנאים שונים המגדירים כל אחד דרך שונה ליצירת עצם מהמחלקה וקביעת מצבו ההתחלתי.

### ו. הפניות

עד כה הכרנו משתנים שיכולים להכיל ערכים פשוטים, כגון מספרים או ערכי אמת. כעת נכיר משתנה שיכול להכיל גם עצם, או למעשה הפניה (reference) אל העצם. כאשר מוגדר משתנה מטיפוס מחלקה, הוא יכול להכיל הפניות לעצמים שהם מופעים של המחלקה, כלומר עצמים מטיפוס המחלקה.

לדוגמה, ההוראה הבאה מופיעה בפעולה הראשית:

Card c1;

ומגדירה משתנהc1 מטיפוס Card. בשלב זה אין ל-c1 ערך ועלינו לאתחל אותו.

בעת זימון הבנאי נוצר מופע של המחלקה Card. הבנאי מחזיר את ההפניה של העצם ואופרטור ההשמה שומר את ההפניה שהוחזרה במשתנה c1:

c1 = new Card("red");

נניח שהוגרל המספר 11

עכשיו c1 מכיל הפניה לעצם מטיפוס Card:

C1:Card Card

אפשר לאחד את הגדרת המשתנה ואת הצבת ההפניה לעצם החדש שנוצר במשתנה זה, ולכתוב:

Card c1 = new Card ("red");

ההכרזה באגף שמאל מגדירה משתנה מטיפוס Card.

באגף ימין נוצר עצם מטיפוס Card שמצבו נקבע לפי ערכי הפרמטרים. באגף זה Card הוא שם הבנאי. הוראה זו מגדירה משתנה ועצם במצב מסוים, ומציבה במשתנה את ההפניה לעצם.



משימה 4: יצירת מספר מופעים

צרו בפעולה הראשית שני מופעים של קלף.

פתרון:

לפניכם אינטראקציה המדגימה כיצד ליצור בפעולה הראשית מספר מופעים של Card:

public class Test

{

public static void main(String[] args)

{

Card c1 = new Card ("red"); //בהנחה שהוגרל במספר 12

Card c2 = new Card ("black"); //בהנחה שהוגרל המספר 7

}

}

c1:Card

c2:Card

c3:Card

### ז. פעולות נוספות

אחרי שהעצם נוצר הוא יכול לבצע את הפעולות המוגדרות בממשק (פרט לפעולות הבנאי). פעולות יכולות לקבל ערכים כפרמטרים וכן להחזיר ערכים. ערכים אלה יכולים להיות מטיפוסים פשוטים או מטיפוסי מחלקות.

רגע חושבים!

אילו פעולות מאפשרות לברר פרטים על מצבו של קלף?

כאשר רוצים לברר פרטים על מצבו של קלף, משתמשים בפעולות המחזירות את הפרטים המבוקשים – שאילתות. אלה הן הפעולות getNum(), getColor(), getStatus() שבממשק המחלקה Card.

פעולות אלה של הקלף אינן מקבלות פרמטרים, אך למרות זאת חובה לכתוב סוגריים מייד לאחר שם הפעולה. לכל אחת משלוש הפעולות יש ערך החזרה מטיפוס שונה.

זימון פעולה לביצוע על ידי עצם נעשה בעזרת שיטת סימון מיוחדת הנקראת סימון-הנקודה (dot notation). כדי להודיע לעצם להפעיל פעולה, נכתוב את שם ההפניה אל העצם, לאחר מכן נכתוב את שם הפעולה המבוקשת ואחריה נקודה, ובסוף את ערכי הפרמטרים בסוגריים:

Card c1 = new Card ("red") ;

String cardColor = c1.getColor();

בשורה הראשונה יצרנו עצם מטיפוס Card. נניח שהמספר שהוגרל הוא 10. את ההפניה לעצם הצבנו בתוך משתנה מטיפוס Card הנקרא c1. בשורה השנייה פנינו ל-c1 בשיטת סימון-נקודה כדי להפעיל את הפעולה getColor(). פעולה זו מחזירה את צבע הקלף שאותו אנו מציבים במשתנה מטיפוס String.

נזכיר כי הבנאי מוזמן באופן שונה באמצעות המילה השמורה new. זו אינה פעולה שמבצע עצם אלא שמבצעת המחלקה.

נדגים זימון של פעולה המחזירה ערך. הזימון מופיע בתוך הפעולה הראשית:

public static void main(String[] args)

{

Card c1 = new Card("red");

String cardColor = c1.getColor();

System.out.println(cardColor););

}

C#

public static void Main(string[] args)

{

Card c1 = new Card("red");

string cardColor = c1.GetColor();

Console.WriteLine(cardColor););

}

קיים סוג נוסף של פעולות, והוא פעולות המשנות את מצבו של העצם ואינן מחזירות ערך. פעולות אלו נקראות פקודות. כדי לציין שטיפוס ערך החזרה שלהן הוא ריק נכתוב void:

שימו לב למשל כי הפעולה setStatus(…) מוגדרת כפעולת void. על אף שהיא איננה מחזירה כל ערך, היא מקבלת כפרמטר ערך מטיפוס boolean ששמו status, ומשנה את מצב הקלף לערך החדש שמתקבל כפרמטר ("פתוח", "סגור").



##### משימה 4: זימון פעולה:

עקבו אחר התוכנית ורשמו מה יודפס כתוצאה מהרצת הפעולה הראשית?

public static void main(String[] args) {

Card c1 = new Card("black"); //נניח המספר שהוגרל הוא 7

Card c2 = new Card("red"); //נניח שהמספר שהוגרל הוא 12

boolean b1 = c1.isGreater(c2);

boolean b2 = c2.isGreater(c1);

System.out.println(b1);

System.out.println(b2);

}

C#

עקבו אחר התוכנית ורשמו מה יודפס כתוצאה מהרצת הפעולה הראשית?

public static void Main(string[] args) {

Card c1 = new Card("black"); //נניח המספר שהוגרל הוא 7

Card c2 = new Card("red"); //נניח שהמספר שהוגרל הוא 12

bool b1 = c1.IsGreater(c2);

bool b2 = c2.IsGreater(c1);

Console.WriteLine(b1);

Console.WriteLine(b2);

}

}

הפעולה האחרונה בממשק המחלקה קלף, toString(), מחזירה מחרוזת המתארת את הקלף כפי שהמתכנת ניסח.

המחרוזת יכולה להיות ארוכה מאוד ומשולב בה מלל רב, למשל:

"The number on the card is 10, the color is red ,the card is open"

אך היא יכולה להיות גם קצרה בהרבה:

Card : 10 , red, open

על אופן ההצגה מחליט המתכנת הכותב את הגדרת המחלקה. פעולה זו משמשת אותנו לרוב לצורך הדפסת תיאורי עצמים.

זימון הפעולה toString():

public static void main(String[] args) {

Card c1 = new Card("red"); //נניח שהוגרל המספר 10

String str = c1.toString();

System.out.println(str);

}

C#

public static void Main(Sstring[] args) {

Card c1 = new Card("red"); //נניח שהוגרל המספר 10

String str = c1.ToString();

Console.WriteLine(str);

}

שימו לב, ניתן להשתמש בכתיב מקוצר באופן הבא:

public static void main(String[] args) }

Card c1 = new Card("red");

System.out.println(c1);

{

C#

public static void Main(string[] args) }

Card c1 = new Card("red");

Console.WriteLine(c1);

{

כאשר מתבקשת הדפסה של אובייקט, השפה מחפשת את פעולת ה-toString() המוגדרת לגבי אותו עצם ומפעילה אותה, ואין צורך לכתוב במפורש:

System.out.println(c1.toString())

C#

כאשר מתבקשת הדפסה של אובייקט, השפה מחפשת את פעולת ה-WriteLine() המוגדרת לגבי אותו עצם ומפעילה אותה, ואין צורך לכתוב במפורש:

Console.WriteLine(c1.WriteLine())

### ח. אתחול משתנים

רגע חושבים!

מה יהיה הפלט עבור קטע התוכנית הבא?

Card c1;

c1.setStatus (true);

הפקודה הראשונה מגדירה משתנה c1, שעדיין אין לו ערך. הפקודה השנייה היא הוראה לבצע את הפעולה setStatus(…) של העצם שהפניה אליו מאוחסנת במשתנה. כיוון שהעצם כלל לא נוצר, הקומפילציה תיעצר ומקבלים הודעה על שגיאת הידור.

משתנים מקומיים, הן מטיפוס בסיסי והן מטיפוס מחלקה, המוגדרים בתוכנית הראשית או בתוך פעולה, אינם מקבלים ערך ברירת מחדל ראשוני ולכן אנו חייבים לאתחל אותם לערך הרצוי. אם לא נעשה זאת, נקבל הודעה על שגיאת הידור.

לדוגמה:

int num ;

System.out.println(num) ;

C#

int num ;

Console.WriteLine(num) ;

ניגש כעת לכתוב את המחלקה Card על סמך הממשק הנתון. נעשה זאת באמצעות יישום השלבים המתוארים בסעיפים שלהלן.

###### א. הכרזה על מחלקה

הגדרת מחלקה חדשה נעשית באופן הבא:

public class NameOfClass

{

//כאן ייכתב גוף המחלקה

}

C#

public class NameOfClass

{

//כאן ייכתב גוף המחלקה

}

השורה המופיעה בראש ההכרזה נקראת **כותרת המחלקה** (**class header**).

המילה השמורה public פותחת את כותרת המחלקה. זוהי אחת מהרשאות הגישה ש-JAVA מציעה למתכנת. הסבר מפורט יותר על הרשאות גישה ועל מטרת השימוש בהן יופיע בסוף הפרק.

המילה השמורה classמציינת כי זוהי הכרזה על מחלקה. אחריה יופיע שם המחלקה וסוגריים מסולסלים, שבתוכם ייכתב גוף המחלקה.

חובה לקרוא למחלקה בשמו של הטיפוס שאותו היא מגדירה, לדוגמה: class Card.

א.1 מוסכמות הכתיבה

ב-JAVA, כמו בכל שפה, ישנן **מוסכמות** (**conventions**) לגבי סגנון הכתיבה בשפה. מוסכמות אלה הן כללים שהמתכנתים בשפה קיבלו על עצמם ומומלץ לציית להן.

המוסכמות המקובלות באשר לשמות של מחלקות הן:

* שם מחלקה מתחיל באות גדולה, לדוגמה: class **C**ard (המילה Card מתחילה באות הגדולה C).
* קהילת המתכנתים ב-JAVA מעודדת שימוש בשמות בעלי משמעות כדי לאפשר קריאה נוחה של קוד (גם אם הוא מכיל שמות מורכבים וארוכים). לכן שם המחלקה ייכתב כך שכל מילה חדשה תתחיל באות גדולה. לדוגמה, שמה של המחלקה "תולעת ורודה שלה נקודות כחולות" ייכתב כך:

class PinkWormWithBlueDots

C#

class PinkWormWithBlueDots

במהלך פרק זה תכירו עוד מוסכמות.

התוכנית יכולה לפעול היטב גם אם חורגים מהמוסכמות, אך למוסכמות חשיבות רבה כדי ליצור קוד מובן, קריא ונוח לשימוש. בתהליך עבודה שלו שותפים אנשים רבים, השימוש במוסכמות אלה יכול ליצור הבדל משמעותי באיכות תהליך העבודה.

א.2 קובץ חדש למחלקה החדשה

את המחלקה החדשה נשמור בקובץ הנקרא Card.java.

כפי שכבר ראינו, כל מחלקה ב-JAVA המוגדרת כבעלת הרשאת גישה פומבית חייבת להישמר בקובץ נפרד ששמו זהה לשם המחלקה, בתוספת הסיומת '.java'. שימו לב, קביעת שם הקובץ בצורה זו היא כלל הכרחי בשפה. ההתאמה בין שמות הקבצים לשמות המחלקות היא שמאפשרת למהדר לקשר ביניהם, ולמצוא באיזה קובץ שמורה כל מחלקה.

###### ב. מצב של עצם

לכל המופעים של מחלקה מסוימת יש את אותם אפיונים (שהם התכונות – attributes), אך הם נבדלים זה מזה במצבם, כלומר בערכי האפיונים שלהם. כדי לייצג מצב של עצם יש לבחור תכונות שיוגדרו במחלקה. קיימת תלות בין הפעולות שעצם יכול לבצע לבין הייצוג שנבחר עבורו, ולכן לרוב בוחרים את התכונות מתוך ידיעת השירותים שהעצם נדרש לספק.

המחלקה קלף (Card) שבה עסקנו בפירוט בחלקו הראשון של הפרק הגדירה טיפוס של קלפים המאופיין בשלוש תכונות: מספר, צבע, ומצב פתוח או סגור. כל אחד ממופעי המחלקה הוא בעל תכונות אלה, ומצבו נקבע על סמך הערכים שלהן. מצבם של מופעים יכול להיות שונה אם ערכי תכונותיהם שונים זה מזה, אולם גם אם כל ערכי התכונות שווים ומצב המופעים זהה לכאורה, עדיין כל מופע הוא עצם נפרד.

ב.1 בחירת תכונות

התכונות (attributes) יהיו הדבר הראשון שאותו מגדירים בכל מחלקה, ושלב זה נקרא ייצוג המחלקה. כדי לייצג את המחלקה יש להחליט מהן התכונות הנדרשות בהתאם לבעיה שמנסים לפתור. התכונות צריכות לייצג את האפיונים החשובים לנו לגבי מצב העצם. בבואנו לכתוב את המחלקה קלף אנו צריכים לאפיין קלף בעזרת תכונות מסוימות. קלפים נבדלים זה מזה במספרם, ולכן סביר שתהיה לקלף תכונה בשם "מספר".

קלפים נבדלים גם בצבעם, במידות האורך והרוחב שלהם וכדומה. בחירת התכונות לייצוג עצמים בתוכנית היא תהליך של הפשטה: אנו מייצגים ישויות תוך הדגשת תכונות מסוימות והתעלמות מאחרות שאינן נראות לנו חשובות ליישום שאנו בונים. לכן לתוכנית המסוימת שבה אנו עוסקים נבחר רק את תכונות הקלף הנחוצות לנו. אם למשל התוכנית עוסקת בבניית מגדל קלפים וחישוב גובהו, אז ממדי הקלף (אורך ורוחב) הן תכונות משמעותית ונחוצות.

ביצירת מופע של מחלקה (עצם מטיפוס המחלקה) התכונות שלו מקבלות ערכים מסוימים, והפעולות שנבצע במהלך התוכנית יכולות לשנות ערכים אלה. ייתכן שיופיעו בממשק פעולות הקובעות את ערכיה של תכונה או מחזירות אותן (פקודות או שאילתות בהתאמה) עבור כל תכונה שנבחרה.

תכונות הן המשתנים הפנימיים של העצם הקובעים את מצבו, ולמעשה את דרך ייצוג המחלקה. את הייצוג הזה נהוג להשאיר לשימוש המחלקה עצמה ולהסתירו מהמשתמשים במחלקה בעזרת הגדרת התכונות כ**פרטיות** (private).

**ב.2 הצהרה על תכונות**

public class Card {

// תכונות (פרטיות)

private int number;

private String color;

private boolean status;

}

C#  
public class Card {

// תכונות (פרטיות)

private int number;

private string color;

private bool status;

}

טיפוס התכונה המייצגת את מספר הקלף הוא int, מספר שלם. שם התכונה המייצגת את מספר הקלף הוא number. טיפוס התכונה המייצגת את צבע הקלף הוא מחרוזתString ושמה הוא color.

המוסכמה הנהוגה לגבי שמות התכונות היא אותה מוסכמה עבור שמות משתנים בכלל: השם ייכתב באותיות קטנות, למעט האות הראשונה של כל מילה פנימית חדשה שתיכתב באות גדולה – למשל, מספר תעודת זהות ייכתב idNumber.

###### ג. בנאי המחלקה – constructor

רגע חושבים!

איך פועל הזיכרון של המחשב ברגע יצירת עצם חדש?

כפי שהבנו עד כה, מחלקה משמשת תבנית שממנה ניתן ליצור עצמים. כדי לעשות זאת צריך להקצות זיכרון עבור העצם ולאתחל את תכונותיו. שלבים אלה מתבצעים בעת זימון בנאי המחלקה.

**ג.1 זימון הבנאי**

בפרק הקודם למדנו שיצירת עצם חדש מתבצעת בעזרת המילה השמורה new. העצם החדש מאותחל מייד לאחר מכן, בעת זימון הבנאי. למשל, כדי ליצור קלף שצבעו אדום ולהציבו במשתנה, נכתוב:

Card c1 = new Card ("red");

מה קורה בפועל?

הפקודה new באגף ימין מקצה מקום בזיכרון של המחשב עבור עצם בעל שלוש תכונות: number מטיפוס int, color מטיפוס String, ותכונה נוספת status מטיפוס boolean כפי שמפורט במחלקה. בשלב זה מאותחלות התכונות על פי ערכי ברירת המחדל של הטיפוסים השונים. לאחר מכן הבנאי מציב ערכים לתכונות ומחזיר הפניה אל העצם שנוצר ואותחל. בשלב האחרון אופרטור ההשמה מציב הפניה זו לתוך המשתנה מטיפוס המחלקה שהוצהר באגף שמאל.

אם הבנאי אינו מציב באופן מפורש ערכים בתכונות של העצם, התכונות נשארות מאותחלות לערכי ברירת המחדל של JAVA. רצוי שלא להסתמך על ערכים אלה אלא לאתחל כל תכונה באופן מפורש.

ג.2 כותרת הבנאי

ב-JAVA אין מילה שמורה המורה על כך שפעולה מסוימת היא בנאי, אך יש מבנה כותרת מיוחד שבעזרתו המהדר מזהה בנאי. **שימו לב** שהמבנה המיוחד של הכותרת הוא הדרך היחידה לזהות בנאי, ולכן חשוב מאוד להקפיד עליו.

מבנה הכותרת של בנאי ייראה כך:

1. שם הפעולה, הזהה לשם המחלקה.
2. רשימת הפרמטרים של הפעולה, בסוגריים עגולים.

את הכותרת מקדימה הרשאת הגישה (על פי רוב public).

public Card (String color)

הרשאת

גישה

רשימת הפרמטרים

שם הפעולה

מהו ההיגיון המצוי בבסיס מבנה זה?

* הרשאת גישה מסוג public מאפשרת למחלקות אחרות להשתמש בבנאי וליצור עצמים מטיפוס המחלקה.
* הזהות בין שם הפעולה לשם המחלקה מאפשרת למהדר לדעת באיזו מחלקה מדובר, וכך להבין מהי תבנית העצם הנדרשת לבנייה ומהו שטח הזיכרון הדרוש. כמו כן, זיהוי המחלקה מאפשר למהדר למצוא את הבנאי בהגדרת המחלקה.
* לבסוף, כמו בכל פעולה, תיסגר כותרת הפעולה בסוגריים עגולים שיכילו את הפרמטרים הדרושים לביצועה (אם יש כאלה) או יישארו ריקים (אם אין).

ג.3 מימוש הבנאי

כדי לאתחל קלף חדש צריך לקבוע ערך התחלתי לתכונות שלו. החלטנו שבבנייה של קלף חדש הוא יהיה תמיד סגור, ומספר הקלף ייקבע בהגרלה, ולכן לפעולה יש פרמטר יחיד – קביעת צבע הקלף בלבד.

**עיינו** בקוד הבנאי של המחלקה Card:

public Card(String color) {

Random rnd = new Random();

this.number = rnd.nextInt(13) + 1;

this.color = color;

this.status = false;

}

C#

public Card(string color) {

Random rnd = new Random();

this.number = rnd.Next(13) + 1;

this.color = color;

this.status = false;

}

}

הפניה לעצם בתוך קוד המחלקה עצמה נעשית בעזרת המילה השמורה this. מילה זו מציינת את העצם **הנוכחי**. במקרה של הבנאי, זהו העצם שזה עתה נבנה.

כיצד מבדילים בין הפרמטרים של הבנאי לבין תכונות העצם אם שמותיהם זהים? במקרה כזה ברירת המחדל תפנה אל הפרמטר של הבנאי, ולא אל התכונה של העצם. כדי לשמור את ערך הפרמטר בתוך התכונה של העצם נשתמש במילה this המתייחסת לתכונת העצם.

בבלוק הפקודות של הבנאי מוגרל מספר בין 1 ל-13, ולאחר מכן הוא מושם בתכונה number של העצם. כמו כן מושם הערךcolor (שהתקבל כפרמטר) בתכונה color של העצם והערך false בתכונה status של העצם.

מעכשיו והלאה, הערכים האלה שמורים בתכונות העצם וקובעים את מצבו. יתר הפעולות יכולות לגשת אל ערכים אלה, להשתמש בהם או לשנות אותם לפי הצורך.

במימוש של בנאי לעולם לא תופיע ההוראה return, כיוון שאין כאן יצירה והחזרה של עצם חדש, אלא אתחול מצב העצם שנוצר בעת ביצוע הפקודה new.

הערה: בדוגמה שראינו, שם הפרמטר זהה לשם התכונה שהוא מאתחל. אנו כותבים זאת כך רק כדי להמחיש את הקשר בין הערך הנשלח לשם התכונה שאותה הוא מאתחל, אך אין זו חובה: שם הפרמטר אינו חייב להיות זהה לשם התכונה.

ג.4 בנאי ללא פרמטרים

ראינו כי פעולת בנאי, בדומה לכל פעולה, יכולה לקבל פרמטרים. אך לעיתים נרצה לכתוב בנאי שאינו מקבל פרמטרים. כותרתו של בנאי כזה תיראה כך:

public Card()

פעולה זו יכולה לאתחל את התכונות על פי קבועים שאינם מתקבלים מהמשתמש, כך:

public Card() {

this.number = 1;

this.color = "black";

this.status = false;

}

}

C#

public Card() {

this.number = 1;

this.color = "black";

this.status = false;

}

אך היא יכולה גם שלא לאתחל במפורש את התכונות, באופן הזה:

public Card()

{

}

C#

public Card()

{

}

במקרה זה יאותחלו התכונות לפי ערכי ברירת המחדל של JAVA. עבור המחלקה קלף, המספר יאותחל ל-0 הצבע יאותחל ל-null והמצב יאותחל לשקר. בנאי כזה ייצור קלף מעוות שמספרו 0 והצבע שלו לא קיים.

כדי למנוע יצירות מעוותות כאלה עדיף ומומלץ לא לסמוך על ערכי ברירת המחדל של JAVA, אלא לאתחל תמיד את הערכים של התכונות באופן מפורש.

בהמשך הפרק נלמד כיצד להגדיר כמה בנאים במחלקה אחת.

**ג.5 בנאי ברירת מחדל**

ב-JAVA קיים מנגנון הדואג לכך שבכל מחלקה יופיע בנאי, גם אם המתכנת שכח להוסיפו או נמנע מכך מסיבה כלשהי. במקרה כזה המנגנון מוסיף למחלקה בנאי ללא פרמטרים שנקרא **בנאי ברירת מחדל** (**default constructor**). הבנאי משאיר את ערכי ברירת המחדל של תכונות העצם כפי שנקבעו על ידי JAVA. מהלך זה אינו מומלץ, כפי שציינו לעיל.

בנאי ברירת מחדל מתווסף למחלקה באופן אוטומטי רק כאשר המתכנת לא הגדיר במחלקה אף בנאי אחר. את הבנאי הזה אי אפשר לראות במפורש בקוד המחלקה, אולם הוא מהווה חלק מממשק המחלקה, ומחלקות אחרות יכולות להשתמש בו כדי ליצור עצמים חדשים.

מנגנון ההוספה האוטומטי של בנאי מונע קיום מחלקות שאין אפשרות לייצר מהן עצמים, אולם כאמור מומלץ לא לסמוך על מנגנון זה ולכלול בכל מחלקה לפחות בנאי אחד (או יותר, כפי שנראה בהמשך) העונה על צורכי המחלקה.

###### ד. תרשימים

ד.1 תרשימים של עצמים

בעת יצירת עצם הוקצה מקום בזיכרון לתכונות שהוגדרו עבורו, ותכונות אלו קיבלו ערכים התחלתיים שקבעו את מצב העצם. כעת נציג תרשים שיתאר באופן חזותי את מצב העצם לאחר היווצרותו.

נדגים זאת עם הפקודה הבאה:

Card c1 = new Card ("black");// נניח שהוגרל המספר 4

הפקודה הקצתה שטח זיכרון לעצם מטיפוס קלף. לעצם שלש תכונות שערכיהן נקבעו בעת הקריאה לבנאי וההפניה אליו. ההפניה מוצבת במשתנה c1. כל המידע הזה מתומצת בתרשים העצם הזה:

number

**4**

colour

c1

#### **Card**

**0**

תרשימים נוספים שישמשו אותנו רבות ביחידה זו הם תרשימי UML לתיאור

מחלקות, כפי שנראה בסעיף הבא.

ד.2 תרשימי UML

דרך מקובלת להצגה תמציתית של מחלקות היא בעזרת שפת המידול **UML** (Unified Modeling Language). התרשים הבא שמציג את המחלקה Card מחולק לשלושה חלקים: שם המחלקה, תכונות ופעולות. בהמשך היחידה נציג פעמים רבות את תרשימי ה-UML ונחסוך בהסברים מילוליים ארוכים.

**רגע חושבים!**

הסתכלו על מחלקת Organ בפרק 1. האם היא מוצגת על פי שפת מידול UML?

**שם המחלקה**

#### תכונות

#### פעולות

int number

String color

boolean status

boolean isOpen

# Card

Card (String color)

int getNumber()

String getColor ()

boolean getStatus()

void setStatus (boolean status)

boolean isGreater(Card other)

String toString()

###### ה. פעולות נוספות

בנוסף לבנאי אחד או יותר מוגדרות במחלקה פעולות נוספות שמופעלות על ידי עצמים הנוצרים בה, ולרוב הן משתמשות בתכונות העצמים. ניזכר כי הפניה לעצמים בתוך קוד המחלקה נעשית באמצעות המילה השמורה thisהמציינת את העצם הנוכחי המבצע את הפעולות.

**שאילתה המאחזרת את מספר הקלף:**

public int getNumber() {

return this.number;

}

C#

public int GetNumber() {

return this.number;

}

פעולה זו מחזירה את מספר הקלף הנוכחי.

**שאילתה המאחזרת את צבע הקלף:**

public String getColor() {

return this.color;

}

C#

public string GetColor() {

return this.color;

}

פעולה זו מחזירה את צבע הקלף הנוכחי.

**שאילתה המאחזרת את מצב הקלף:**

public boolean getStatus() {

return this.status;

}

C#

public bool GetStatus() {

return this.status;

}

פעולה זו בודקת האם הקלף פתוח או סגור , כלומר האם ערך התכונה status של הקלף הנוכחי הוא אמת או שקר. הפעולה מחזירה ערך בוליאני "אמת" אם הקלף פתוח, ו"שקר" אם הקלף סגור.

**פקודה המשנה את מצב הקלף:**

public void setStatus(boolean status) {

this.status = status;

}

C#

public void SetStatus(bool status) {

this.status = status;

}

פעולה זו משנה את מצב הקלף לפתוח או סגור בהתאם לפרמטר המתקבל.

**שאילתה המשווה בין שני קלפים:**

public boolean isGreater(Card other) {

return this.number > other.number;

}

C#

public bool IsGreater(Card other) {

return this.number > other.number;

}

פעולה זו בודקת האם המספר של הקלף הנוכחי גדול מערכו של המספר בקלף other.

הפעולה מחזירה ערך בוליאני "אמת" אם ערך התכונה number בקלף הנוכחי גדול מערך התכונה number בקלףother , אחרת הפעולה מחזירה "שקר".

**שאילתה המאחזרת תיאור של קלף:**

פעולה זו בונה מחרוזת המתארת את העצם ומחזירה אותה. המחרוזת מכילה מידע שהמתכנתים מעוניינים להציג לגבי העצם. בדרך כלל יהיו אלה ערכי כל התכונות של העצם או רק חלק מהן:

public String toString() {

String str = "Card " + this.number + " " + this.color + " ";

if (this.status == true)

str = str + "open";

else

str = str + "close";

return str;

}

C#

public override string ToString() {

string str = "Card " + this.number + " " + this.color + " ";

if (this.status == true)

str = str + " open";

else

str = str + " close";

return str;

}

הפעולה יצרה מחרוזת המכילה שרשור של ערכי תכונות הקלף בצירוף שם המחלקה. בשורה האחרונה הוחזרה המחרוזת כערך ההחזרה של הפעולה.



משימה לסיכום ביניים:

נדגים את השימוש בפעולות שסקרנו עד לכאן:

public static void main(String[] args)}

Card c1 = new Card("red"); //נניח שהוגרל המספר 5

Card c2 = new Card("black"); //נניח שהוגרל המספר 11

Card c3 = new Card("red"); //נניח שהוגרל המספר 11

c1.setStatus(true);

c2.setStatus(false);

boolean check1 = c2.isGreater(c1);

System.out.println(c1);

System.out.println(c2);

boolean check2 = c2.isGreater(c3);

System.out.println(c2);

System.out.println(c3);

א. תארו את מצב הקלפים בעזרת תרשימי עצמים

ב. השיבו מהו הפלט של התוכנית?

C#

public static void Main(string[] args)}

Card c1 = new Card("red"); //נניח שהוגרל המספר 5

Card c2 = new Card("black"); //נניח שהוגרל המספר 11

Card c3 = new Card("red"); //נניח שהוגרל המספר 11

c1.SetStatus(true);

c2.SetStatus(false);

bool check1 = c2.IsGreater(c1);

Console.WriteLine(c1);

Console.WriteLine(c2);

bool check2 = c2.IsGreater(c3);

Console.WriteLine(c2);

Console.WriteLine(c3);

}

פתרון

מצב הקלפים בתום שתי הפעולות הראשונות:

capacity

**5**

currentAmount

b1

#### **Bucket**

**0**

capacity

**4**

currentAmount

b2

#### **Bucket**

**0**

מצב הקלפים בתום שתי הפעולות הבאות:

capacity

**5**

currentAmount

b1

#### **Bucket**

**3**

capacity

**4**

currentAmount

b2

#### **Bucket**

**4**

מצב הקלפים בתום הפעולה האחרונה

capacity

**5**

currentAmount

b1

#### **Bucket**

**5**

capacity

**4**

currentAmount

b2

#### **Bucket**

**2**

לסיום התבוננו במחלקה Card בשלמותה:

public class Card {

private int number;

private String color;

private boolean status;

public Card(String color) {

Random rnd = new Random();

this.number = rnd.nextInt(13) + 1;

this.color = color;

this.status = false;

}

public int getNumber() {

return this.number;

}

public String getColor() {

return this.color;

}

public boolean getStatus() {

return this.status;

}

public void setStatus(Boolean status) {

this.status = status;

}

public boolean isGreater(Card other) {

return this.number > other.number;

}

public String toString() {

String str = "Card " + this.number + " " + this.color + " ";

if (this.status == true)

str = str + "open";

else

str = str + "close";

return str;

}

}

C#

public class Card {

private int number;

private string color;

private bool status;

public Card(string color) {

Random rnd = new Random();

this.number = rnd.Next(13) + 1;

this.color = color;

this.status = false;

}

public int GetNumber() {

return this.number;

}

public String GetColor() {

return this.color;

}

public bool GetStatus() {

return this.status;

}

public void SetStatus(bool status) {

this.status = status;

}

public bool IsGreater(Card other) {

return this.number > other.number;

}

public override string ToString() {

String str = "Card " + this.number + " " + this.color + " ";

if (this.status == true)

str = str + "open";

else

str = str + "close";

return str;

}

}

##### תרגילים פרק 6 חלק א'

### משימה 1: נקודת התחלה

נקודה במישור Point מוגדרת כבעלת שתי קואורדינטות : x ו-y

לפניכם ממשק המחלקה Point:

|  |  |
| --- | --- |
| בנאי של המחלקה Point. הפעולה יוצרת נקודה חדשה על פי ערכי הפרמטרים. | Point (double x, double y) |
| שאילתה המחזירה את קואורדינטת ה-x של הנקודה. | double getX() |
| פקודה המקבלת ערך מטיפוס **double** , וקובעת את קואורדינטת ה-x של הנקודה בהתאם. | void setX (double x) |
| שאילתה המחזירה את קואורדינטת ה-y של הנקודה. | double getY() |
| פקודה המקבלת ערך מטיפוס **double**, וקובעת את קואורדינטת ה-y של הנקודה בהתאם. | void setY (double y) |
| שאילתה המחזירה מחרוזת המתארת את נתוני הנקודה על פי הצורה הבאה:  ( <X> , <Y> ) | String toString() |

מה עליכם לעשות?

1. יצגו את המחלקה Point (כלומר קבעו מה יהיו התכונות של מופעי המחלקה).
2. ציירו תרשים UML המתאים למחלקה.
3. צרו מחלקה חדשה בשם Point. שימו לב! שם המחלקה ושם הקובץ שבו היא נמצאת חייבים להיות זהים.
4. ממשו את פעולות המחלקה Point.

כדי לבדוק שהמחלקה שכתבתם עובדת כראוי, עליכם לכתוב תוכנית בדיקה לפי ההנחיות הבאות:

המחלקה TestPoint

ד . צרו מחלקה נוספת בשם TestPoint (בקובץ TestPoint.java).

בתוך המחלקה TestPoint כתבו פעולת main(...) המבצעת את משימות האלה:

* + בונה נקודה חדשה לפי הקואודינטות (7 ,43).
  + בונה נקודה חדשה נוספת לפי הקואורדינטות (5 ,5).
  + מדפיסה את שתי הנקודות בעזרת המחרוזת המוחזרת מהפעולהtoString() כך (אין צורך לכתוב את שם הפעולה במפורש):

System.out.println (xxx);

* + מחליפה בין קואורדינטות ה-x של שתי הנקודות, תוך שימוש בפעולות השונות של המחלקה.
  + מדפיסה שוב את הנקודות החדשות.



### משימה 2: משחק בקוביות

ממשק המחלקה קוביה – Die

המחלקה Die (קובייה) מגדירה קובייה שלה 6 פאות. על הפאות מופיעים המספרים 1 עד 6. כאשר הקובייה נמצאת במנוחה, ונשאלת השאלה "מהו המספר שהקובייה מראה?" התשובה לכך היא: "המספר שנמצא על הפאה העליונה".

|  |  |
| --- | --- |
| Die() | בנאי של המחלקה קובייה. הפעולה יוצרת עצם מטיפוס Die.  הקובייה שנוצרה מראה מספר אקראי בין 1 ל-6. |
| void roll() | פקודה המדמה "הטלת קובייה". בתום הפעולה מתעדכן המספר שהקובייה מראה לאחר ההטלה. |
| int getNum() | שאילתה המחזירה את המספר שמראה הקובייה. |

מה עליכם לעשות?

1. חִשבו מהן התכונות הנחוצות למחלקה Die וציירו UML מתאים למחלקה.
2. כתבו את המחלקה Die במלואה.

**רמז**: כדי להטיל את הקוביות באופן אקראי השתמשו בפעולה random() של המחלקה Math. על אופן פעולתה ראו ב-Java API.

1. כתבו מחלקה בשם DiceGame (משחק קוביות), ובה פעולה ראשית היוצרת שתי קוביות.

בכל תור תטיל התוכנית את שתי הקוביות עד אשר יתקבל הצירוף: 6, 6. בכל תור יש למעשה שתי הטלות של שתי קוביות המשחק.

1. התוכנית תדפיס את תוצאות ההטלות בכל התורות. (תורים ? )
2. כאשר יתקבל הצירוף 6, 6 תיעצר התוכנית ותדפיס כמה תורות התקיימו עד אשר קיבלנו 6, 6.

### משימה 3: תיעוד Javadoc

רקע

בדף עבודה זה תלמדו 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.

ההערות המיועדות להפקת דפי תיעוד חיצוניים מכילות נתונים שונים שמהם המהדר מתעלם, ומנגנון התיעוד משתמש בהם כדי לתרגם אותם למידע משמעותי בדף הממשק. כך למשל, כל פרמטר הנשלח לפעולה וכל ערך החזרה של פעולה יפורטו בדף הממשק. בשלב התיעוד יופיע בהערה סימן @ ולידו הסבר לגבי מהות הפרמטר או ערך ההחזרה. בדף הממשק יתקבלו מסימון זה שורות מידע מסודרות.

מה עליכם לעשות?

עליכם לתעד את הקוד של המחלקה Point שכתבתם בדף העבודה הקודם, תוך שימוש בתיעוד האוטומטי של סביבת ה-Eclipse. לאחר שהגדרתם כותרת של פעולה, עלו עם הסמן לשורה **שמעל** לכותרת הפעולה (ניתן לבצע אותו תהליך גם על כותרת של מחלקה), כתבו \*\*/ והקלידו Enter (שימו לב שעליכם להקליד **שתי** כוכביות). באופן אוטומטי תופיע מעל כותרת הפעולה הערה המסתיימת ב-/\*. בתוכה יופיעו סימני ה- Javadoc הרלוונטיים:

@author username – אם זו כותרת מחלקה

@returns – אם הפעולה אינה **void**

@param – אם יש פרמטרים

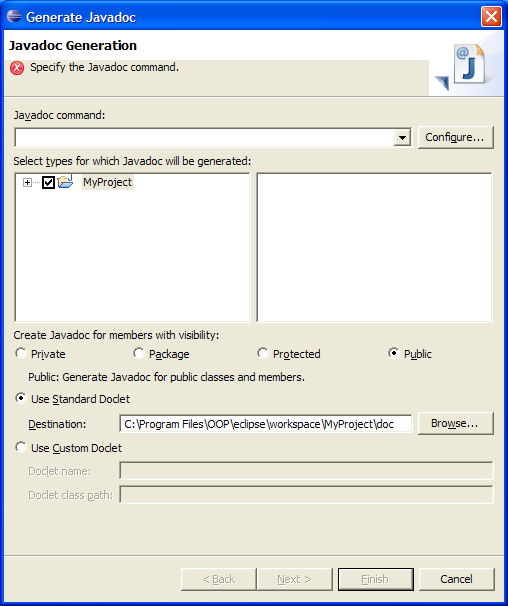
וכו'.

השלימו את התיעוד הנדרש ליד כל סימן @.

**הערה**: ביכולתכם לתעד כל מחלקה אחרת שכתבתם, אך השתדלו שזו תכיל מגוון פעולות ולא רק פעולת main, כך שתוכלו להתרשם מהממשק המתקבל.

**הפעלת מנגנון התיעוד**

1. ברשימת התפריטים בחלק העליון של חלון סביבת העבודה יש לבחור את תפריט Project, ובתוכו את Generate Javadoc.
2. בחרו בהמשך המסך את הפרויקט שעבורו אתם רוצים ליצור את התיעוד על ידי סימונו במקום המתאים. אם ברצונכם לייצר ממשק רק לחלק ממחלקות הפרויקט (לדוגמה, אם אתם רוצים לוותר על יצירת תיעוד לספריות), ניתן ללחוץ על Next ולבחור ספציפית עבור אילו מחלקות ייווצר Javadoc.



 כברירת מחדל – קובצי ה-HTML של ה-Javadoc ייווצרו תחת ספריית הפרויקט בתת-ספרייה בשם doc.

1. אישור הביצוע בשלב זה ימקם את קובץ התיעוד באותה ספרייה שבה שמור הפרויקט שלכם, בתת-ספרייה בשם doc. אם ברצונכם לשמור את התיעוד במקום אחר – בחרו את המיקום בשלב זה.
2. לאחר אישור הפעולה ידווח המהדר דיווחים שונים בחלון ה-Console. אל תיבהלו, הרשימה ארוכה, אך בסוף התהליך יתוספו לפרויקט שלכם בחלון ה-Package Explorer תיקיות התיעוד של הפרויקט. פתחו את תיקיית ה-doc וחפשו את הקובץ הנושא את שם המחלקה שבה אתם מעוניינים. סיומת הקובץ תהיה .html פתחו את הקובץ, עברו עליו והשוו בין התיעוד שנוצר לתיעוד שכתבתם אתם במחלקה.

**סימני תיעוד**

למנגנון התיעוד אפשרויות רבות, ואנו נציג בפירוט שלוש מתוכן שמתוספות להערות באופן אוטומטי:

* 1. **שם כותב המחלקה** – בתוך גוף ההערה המתעדת את המחלקה יתוסף הכיתוב@author <yourName>. לאחר הכיתוב @author עליכם להוסיף את שמכם.

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 במקום המתאים.

* 1. **פרמטרים של פעולה** – מנגנון התיעוד ישלוף מתוך הסוגריים של הפעולה את רשימת הפרמטרים הנשלחים אליה, ויציין אותם בעזרת @param בתוך גוף ההערה. כדי להוסיף הסבר למהות הפרמטרים, השלימו את ההסבר בהמשך לסימון.

לדוגמה, את הפעולה:**void** setX(**double** x)

נוכל לתעד כך:

/\*\*

\* changes the x coordinate to the given value.

\* @param x the new value for the x coordinate.

\*/

* 1. **ערך החזרה** –כדי להוסיף את פירוט ערך ההחזרה של פעולה, נשתמש בפקודה @return.

לדוגמה, את הפעולה: **double** getY()

נוכל לתעד כך:

/\*\*

\* returns the y coordinate.

\* @return the y coordinate.

\*/

גם בשני מקרים אלה ישלוף מנגנון התיעוד את האינפורמציה שמופיעה לצד ה-@ וימקם אותה במקום הנכון בדף הממשק. בדקו!

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.

**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](http://www.java.sun.com). כדי להוסיף תגית חדשה עליכם להקליד @ בתוך ההערה ולבחור את התגית הרצויה מתוך חלון האפשרויות שנפתח.

|  |  |
| --- | --- |
| **התגית** | **משמעות התגית** |
| @author | שם מחבר המחלקה |
| @param | פרמטר המועבר לשיטה |
| @return | הערך שמחזירה השיטה |
| @see | הפניה למחלקה אחרת הקשורה למחלקה זו |
| @version | גרסת המחלקה |

פרק 6 המשך – נושאים מתקדמים

###### ו. העמסת פעולות



משימה 5: העמסת פעולות

מתכנן המחלקה card החליט שהוא רוצה להוסיף למחלקה פעולה שמשנה את מצב הקלף כך שאם הקלף היה פתוח הוא ייסגר, ואם היה סגור הוא ייפתח.

לפניכם כותרת הפעולה כפי שהציג המתכנן:

public void setStatus()

האם לדעתכם ניתן להוסיף את הפעולה לממשק המחלקה, למרות שיש כבר במחלקה פעולה בעלת שם זהה:

public void setStatus(boolean status)

**פתרון:**

ניתן לכתוב באותה המחלקה כמה פעולות בעלות אותו השם, בתנאי שרשימת הפרמטרים שלהן שונה. השונות יכולה להיות במספר הפרמטרים, בטיפוסיהם או בסדר הופעת הפרמטרים בסוגריים.

לכן ניתן להוסיף את הפעולה setStatus() ללא פרמטרים למרות שיש כבר במחלקה פעולה setStatus(boolean status) עם פרמטר מטיפוס בוליאני.

כאשר יש הבדל במספר הפרמטרים (כמו במקרה זה) או בסדר הופעתם, המהדר יודע לבחור את הפעולה הנכונה עבור כל זימון. למנגנון המאפשר להגדיר כמה פעולות בעלות אותו השם הנבדלות זו מזו ברשימת הפרמטרים שלהן קוראים **העמסה** (**overloading**). שימו לב, השוני בין הכותרות של שתי הפעולות חייב להיות ברשימת הפרמטרים – במספרם או בסדרם. שוני בטיפוס ערך ההחזרה אינו אפשרי.

נדגים זאת באמצעות הפעולה:

public int setStatus(boolean status)

פעולה זו נבדלת מהפעולה הקיימת setStatus(boolean status) רק בטיפוס ערך ההחזרה, ולכן המהדר לא יקבל זאת. כיוון שזימון פעולה מאופיין רק על ידי שם הפעולה והפרמטרים המועברים אליה, המהדר אינו יכול להבדיל בין פעולות שונות בעלות אותו השם ואותם הפרמטרים רק על פי טיפוס ערך החזרה שונה, שכלל אינו כלול בזימון הפעולה.

אין למהדר מספיק מידע כדי להבדיל בין פעולה בעלת ערך החזרה int לבין פעולה בעלת ערך החזרה void, כך שהוא יידע איזו פעולה מבין השתיים מזומנת.

באמצעות מנגנון ההעמסה ניתן לכתוב בנאים שונים בהתאם לרצון המתכנת.

בנוסף לבנאי המוגדר בממשק המחלקה, ניתן להגדיר בנאי בכותרת הבאה:

public Card(int number ,String color)

פעולה זו תבנה קלף שמספרו number והכותרת שלו נקבעת על פי הפרמטר color.

כמו כן ניתן להגדיר בנאי נוסף ללא פרמטרים:

public Card()

כאשר מממשים פעולה זו, המתכנת יכול להחליט על ערכי ברירת מחדל משלו לתכונות.

לדוגמה:

מספר הקלף יהיה תמיד 1 והצבע שלו יהיה תמיד "black".

תזכורת: המהדר יגדיר באופן אוטומטי בנאי ברירת מחדל רק אם במחלקה כלל לא הוגדרו בנאים. לכן רצוי ומקובל שהמתכנת יגדיר לפחות בנאי אחד בכל מחלקה.

ו.1. בנאי מעתיק



משימה 6: העתק הדבק

נתון הקלף c1

Card c1 = new Card("red") ; //נניח שהוגרל המספר 12

נרצה להגדיר קלף חדש c2 שיהיה עותק מדויק של הקלף שהמשתנה c1 מצביע עליו. איך עושים זאת?

פתרון:

int num = c1.getNumber( ) ;

String str = c1.getColor() ;

Card c2 = new Card (str);

לא ניתן לקבוע את מספרו של הקלף החדש, שכן הבנאי לא מאפשר זאת. אז מה הפתרון?

יצירת עותק של עצם היא פעילות חשובה שמתכנתים נדרשים לה לא מעט. זכרו שבנאי מאתחל תכונות של עצם חדש שנוצר. יצירת עותק נעשית בעזרת פעולה המקבלת עצם קיים כפרמטר ומעתיקה את ערכי תכונותיו לעצם חדש. זהו בנאי כיוון שהוא מחזיר עצם חדש, מאותחל, מטיפוס המחלקה. לבנאי שכזה קוראים בנאי מעתיק (copy constructor).

**רגע חושבים!**

כיצד בנאי מעתיק יכולה לשמש במסגרת כתיבה של תוכנית?

מנגנון ההעמסה הוא זה שיאפשר להוסיף לבנאים הקיימים במחלקות גם בנאים מעתיקים זכרו שבתוך המחלקה עצמה ניתן לפנות לתכונות הפרטיות של כל מופע של המחלקה, ולכן הקוד של בנאי מעתיק המצורף למחלקה Card ייראה כך:

public Card(Card other) {

this.number = other.number;

this.color = other.color;

this.status = other.status;

}

C#

public Card(Card other) {

this.number = other.number;

this.color = other.color;

this.status = other.status;

}

כיוון שזהו בנאי, לא יוצרים את העצם החדש בתוך מימוש הפעולה, אלא רק מאתחלים את תכונותיו שם. העצם נוצר באמצעות הפקודה new בעת זימון הפעולה:

Card c1 = new Card("black"); // בהנחה שהוגרל המספר 7

Card b2 = new Card(c1);

בשורה הראשונה נוצר עצם שמספרו 7 וצבעו "black" והוא סגור. בשורה השנייה נוצר עותק של העצם הזה, וההפניה אליו מוצבת ב-c2:

capacity

**7**

currentAmount

c1

#### **Card**

**green**

capacity

**7**

currentAmount

b2

#### **Bucket**

**green**

ז. הכמסה

עיקרון ה**הכמסה** (**encapsulation**) הוא מיסודותיו של תכנות מונחה עצמים. לפי עיקרון זה, עצם דומה לקופסה שחורה המספקת שירותים למשתמשים בה. מי שמשתמש בעצם חייב להכיר את הממשק של הקופסה, כלומר את השירותים שהעצם יודע לספק. לעומת זאת, האופן שבו העצם מממש את השירותים הללו צריך להיות מוחבא בתוך הקופסה השחורה, נסתר מעין המשתמש. הכמסה מבטאת שאיפה לעטוף את המימוש בתוך קפסולה (כמוסה), ומכאן שמה. קל לראות כי עיקרון ההכמסה אינו אלא עיקרון הסתרת המידע, הגורס כי עצם צריך לחשוף בפני עצמים אחרים רק את שירותיו ולהסתיר מהם את יתר הפרטים, את הייצוג ואת המימוש, שאינם חיוניים לצורך השימוש בשירותים האלה.

עיקרון ההכמסה אִפשר לנו בפרקים הקודמים לצייר תרשימים בעזרת הצב בלי לדעת מהן תכונותיו וכיצד הוא מבצע את תזוזותיו. די היה לנו להכיר את ממשק הפעולות כדי להפעיל את הצב כראוי.

בסעיף זה תכירו מנגנון חשוב המאפשר הכמסה ב-JAVA ותלמדו איך משתמשים בו.

ז.1 הרשאות גישה

כאשר אנו מגדירים פעולה או תכונה, ובקיצור – איבר, עלינו להחליט אילו עצמים יוכלו לגשת אליו. **הרשאת גישה** (**access specifier**) נקבעת על פי ההגדרה המופיעה במחלקה כחלק מהגדרת האיבר. ב-JAVA קיימות כמה רמות של הרשאות גישה, בהן:

1. הרשאת גישה **פומבית** (**public**) ***–*** כאשרהגדרת האיבר כוללת את המילה השמורה **public**, עצמים מכל המחלקות ב-JAVA יכולים לגשת אליו.
2. הרשאת גישה **פרטית** (**private**) – כאשר הגדרת האיבר כוללת את המילה השמורה **private**, ניתן לגשת לאיבר רק בקוד המחלקה שבה הוא מוגדר. המהדר יפסול כל גישה לאיבר כזה אם נעשתה מחוץ למחלקה. כל פנייה אל האיבר בתוך המחלקה תהיה חוקית.

מהאמור לעיל נובע שניתן לגשת לאיבר של עצם בשני מקרים:

* אם האיבר מוגדר פומבי.
* אם האיבר מוגדר פרטי והגישה אליו היא מתוך הקוד של המחלקה שהוא מוגדר בה. דוגמה מוכרת לכך היא אתחול תכונות פרטיות בתוך בנאי.

ראו דוגמה של בנאי מעתיק:

public Card(Card other) {

this.number = other.number;

this.color = other.color;

this.status = other.status;

}

C#

public Card(Card other) {

this.number = other.number;

this.color = other.color;

this.status = other.status;

}

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



נתונות שתי מחלקות.

האם המחלקהA עוברת הידור? נמקו תשובתכם.

האם המחלקה B עוברת הידור?

|  |  |
| --- | --- |
| public class A {  private int n;…  public int first() {  return this.n + 1;  }  private int doMore() {  return this.n - 1;  }  } | public class B {  private A obj;  public void doing() {  …  System.out.println((this.obj).first()); //1  System.out.println((this.obj).n); //2  System.out.println((this.obj).doMore()); //3  }  } |

פתרון:

בתוך המחלקה A ניתן לפנות ישירות לתכונה שהוגדרה פרטית.

בתוך המחלקה B לא ניתן לפנות ישירות לתכונה שהוגדרה פרטית במחלקה A או לפעולה שהוגדרה פרטית במסגרת המחלקה A.

לכן הוראה 2 והוראה 3 אינן חוקיות.

ז.2 למה משמשות הרשאות גישה?

גישה מבוקרת לתכונות

מחלקה נבנית כדי לתת תבנית יישומית להגדרה מופשטת שמתכנת יצר עבור יישום כלשהו. כאשר מתכנת מגדיר עצמים שלהם הוא נזקק בתוכניתו, הוא מתחיל בהגדרת טיפוס מופשט, בהגדרת מצבם של מופעי הטיפוס ובהגדרת פעולות ממשק המאפשרות החזרת מידע ושינוי המצב של העצם. על פי ההגדרה של המצב והפעולות, על המתכנת לבחור בייצוג שיאפשר את מימושן של פעולות אלה. כך הוא יוכל לממש את הפעולות על פי הייצוג שבחר. הפעולות הן הדרך לפעול על עצמים ולשנות את מצבם. בדרך כלל לא מקובל לאפשר למשתמש לגשת אל התכונות ישירות, שלא דרך הפעולות.

למשל, למחלקה קלף יש שלוש תכונות: מספר, צבע ומצב. ההיגיון אומר שהתכונה הראשונה אינה ניתנת לשינוי: מספרו של הקלף יישאר כפי שהוא. על מנת לעשות זאת ניתן להגדיר את התכונה הזאת פרטית ולא לכתוב פעולה המאפשרת לשנותה. כך נבטיח שמשתמשי המחלקה לא יוכלו לשנות את ערכו של המספר.

לעומת זאת, תכונת מצב הקלף משתנה במשך ריצת התוכנית כאשר הופכים את הקלף. השינויים בערך התכונה עצמה צריכים להתבצע במגבלות שמציבה התכונה. הגדרת התכונה כפרטית, כך שעדכונה ייעשה רק בעזרת הפעולות setStatus( ….) מבטיחה שהתוכנית תרוץ בלי לחרוג מן המגבלות הנבדקות בקוד הפעולות.

השימוש בהרשאות גישה מאפשר למתכנת המחלקה להבטיח שימוש תקין בעצמים של המחלקה שלו. מי שמשתמש במחלקה מוגבל לפעולות שהוגדרו עבורו, ואין לו גישה לתכונות עצמן.

נציג כלל מקובל וחשוב: על פי רוב, אין לתת למשתמש גישה ישירה לתכונות של העצם. לכן נגדיר את תכונות העצם כפרטיות.

הפרדה בין ממשק למימוש

באמצעות הרשאות גישה יכול כותב המחלקה גם להפריד בקלות בין ממשק המחלקה לבין המימוש שלה. האיברים הפומביים המוגדרים באמצעות ההרשאה public מרכיבים את ממשק המחלקה. עצמים ממחלקות אחרות יכולים לעבוד עם עצם ממחלקה זו ולתקשר עמו רק דרך ממשק זה בזימון פעולות פומביות או בגישה לתכונות פומביות. לעומת זאת, האיברים הפרטיים המוגדרים באמצעות ההרשאה private משמשים רק למימוש המחלקה. זימון פעולות פרטיות או גישה לתכונות פרטיות אפשרי רק מתוך קוד המחלקה.

להפרדה הברורה בין ממשק למימוש יתרון בולט: כיוון שעצמים של מחלקות אחרות אינם יכולים להשתמש באיברים הפרטיים של המחלקה, אנו יכולים לשנות ולשפר את המימוש שלה מעת לעת בלי לפגוע בתוכניות המשתמשות בה, כל עוד הממשק של המחלקה אינו משתנה.

מכיוון שכל הפעולות המופיעות בממשק הן פעולות פומביות, איננו מציינים את הרשאת הגישה public בטבלאות הממשק לאורך היחידה, אבל בקוד המחלקות יש לכלול הרשאת גישה זו.

מלבד הפעולות המופיעות בממשק שבאמצעותן המחלקה מספקת את שירותיה, נמצא פעמים רבות פעולות נוספות המסייעות לכותב המחלקה במימוש הפנימי של המחלקה. פעולות עזר אלה אינן צריכות להופיע בממשק המחלקה, והן יוגדרו פרטיות משום שנועדו לכותבי המחלקה בלבד.

ז.3 טיפוס נתונים מופשט

בתחילת הסעיף ציינו כי עיקרון ההכמסה הוא אחד מעקרונותיו של תכנות מונחה עצמים. עיקרון זה מחייב הפרדה מוחלטת בין ממשק למימוש, ויתרון אחד של הפרדה זו כבר פורט לעיל. כעת נצביע על יתרון חשוב נוסף להפרדה. למדנו שדרך הייצוג של הטיפוסים שמגדירות המחלקות מוסתרת מעיני המשתמשים, והם יכולים להשתמש במופעי המחלקה רק בעזרת פעולות הממשק. לפיכך ניתן להחליף את הייצוג של המחלקה ולהתאים את מימוש פעולותיה לייצוג החדש מבלי שהמשתמשים יצטרכו לשנות את התוכניות שהשתמשו במחלקה זו. יתרה מכך, המשתמשים כלל אינם צריכים לדעת על שינוי הייצוג והמימוש. שינויים אלה יכולים להקל במקרים רבים על תחזוקת מערכות מחשב, על שינוי גרסאות תוכנה ועוד. נדגים בקצרה:

המחלקה Rectangle מגדירה מלבן שצלעותיו מקבילות למערכת הצירים. ניתן לייצג את המחלקה בעזרת ערכי הקואורדינטות של שתי נקודות: הקודקוד הימני התחתון והקודקוד השמאלי העליון של המלבן. כלומר ייצוג המחלקה ייעשה בעזרת ארבע תכונות שערכיהן מספריים:

xRightBottom, yRightBottom, xLeftTop, yLeftTop

הפעולות במחלקה ישתמשו בתכונות אלה לחישובים שונים, כגון חישוב שטח המלבן או היקפו.

אם נחליט לשנות את ייצוג המחלקה לייצוג בעזרת נקודת המרכז של המלבן, אורך המלבן ורוחבו, יהיה עלינו לשנות את כל אופן מימוש הפעולות, שכן הערכים שעליהן יתבססו הפעולות ישתנו. שינויים אלה אינם מעניינים את המשתמש, שכן הפעולות לאחזור שטח מלבן, היקפו ומיקומו במערכת הצירים עדיין קיימות, והתוכנית שלו משתמשת רק בהן. כל עוד מתכנתי המחלקה ישמרו על הממשק, הם יכולים לשנות את הייצוג הפנימי ואת המימוש של הפעולות מבלי שהמשתמשים החיצוניים ייפגעו מכך או אפילו יהיו מודעים לכך.

טיפוס נתונים המוגדר רק באמצעות הפעולות שניתן לבצע על מופעיו נקרא **טיפוס נתונים מופשט** **ADT** (**abstract data type**). מחלקה הממומשת באופן שמסתיר לגמרי את דרך ייצוגה ואת המימוש שלה, וחושף רק את הפעולות שניתן לבצע על העצמים הנוצרים ממנה, היא מחלקה המגדירה טיפוס נתונים מופשט.

בשער הבא נחזור לדון בטיפוסי נתונים מופשטים, נרחיב את הגדרתם ונגלה את יתרונות השימוש בהם.

##### סיכום

* ניתן לראות את המחלקה (class) כתבנית המגדירה טיפוס נתונים חדש שממנה ניתן ליצור מופעים מטיפוס זה.
* מחלקה כוללת הצהרות על תכונות, שהן המשתנים הפנימיים של כל עצם הנוצר ממנה. משתנים אלה הם הייצוג של המחלקה. המשתנים מאותחלים בזמן יצירת העצם, והערכים שבתוכם יכולים להשתנות במהלך התוכנית. ערכי המשתנים מייצגים את מצבו של העצם ברגע נתון. בדרך כלל מוגדרות התכונות כפרטיות.
* השלבים בכתיבת מחלקה חדשה על פי ממשק נתון:

1. כותרת המחלקה.
2. ייצוג המחלקה: בחירת התכונות והטיפוסים שלהן.
3. מימוש פעולות ממשק המחלקה.

* בנאי (constructor) הוא דרך ליצור עצמים (objects) המהווים מופעים שונים של המחלקה. הבנאי מאתחל את תכונות העצם בהתאם לפרמטרים המועברים אליו ובהתאם לשיקול הדעת של המתכנת, או על פי ברירות מחדל.
* בעת זימון הבנאי נוצר מופע של המחלקה. הבנאי מחזיר את ההפניה של העצם ואופרטור ההשמה שומר את ההפניה שהוחזרה במשתנה המוגדר.
* רוב הפעולות המוגדרות במחלקה הן פעולות של העצמים. בעזרתן ניתן לגשת לתכונות, להשתמש בערכיהן ולשנות אותם.
* הרשאות הגישה מאפשרות לקבוע אילו איברים של העצמים יהיו נגישים ואילו לא כאשר נמצאים מחוץ לקוד המחלקה. הרשאות הגישה מאפשרות הפרדה בין הממשק למימוש ותומכות ברעיון ההכמסה. הרשאת הגישה private מאפשרת גישה רק בתוך המחלקה. הרשאת הגישה public מאפשרת גישה גם מחוץ למחלקה.
* ההפרדה בין ממשק (interface) למימוש (implementation) מאפשרת להחליף את ייצוג המחלקה ואת אופן מימושה מבלי שהמשתמשים יהיו מודעים לשינוי. כאשר קיימת הפרדה מלאה נתייחס למחלקה כאל טיפוס נתונים מופשט.
* אפשר לפנות אל הפעולות הפומביות, כולל הפעולה הראשית main(…), מתוך מחלקות אחרות. לעומת זאת, לפעולות הפרטיות אפשר לפנות רק מתוך קוד המחלקה עצמה.
* מנגנון התיעוד Javadoc מאפשר ליישם את העיקרון של עבודה עם ממשקים, שהוא מעמודי התווך של תכנות מונחה עצמים. בעזרת מנגנון זה נוכל לייצר בקלות תיעוד לממשקים בעבור מתכנתים אחרים המשתמשים במחלקות שכתבנו. התיעוד נוצר על פי קוד המחלקה המתועד.
* מנגנון ההעמסה (overloading) מאפשר להגדיר במחלקה אחת פעולות בעלות אותו השם אך רשימת פרמטרים שונה. אחת הפעולות השימושיות תהיה בנאי מעתיק (copy constructor) המשמש לשכפול עצמים.
* פעולות פנימיות הן פעולות המופיעות בממשק מחלקה. פעולות חיצוניות הן פעולות המקבלות פרמטר מטיפוס המחלקה אך מוגדרות מחוץ למחלקה עצמה.
* דרך נוחה להצגת מחלקות היא תרשימי UML. דרך נוחה להצגת עצמים היא בעזרת תרשימי עצמים.

מושגים

|  |  |
| --- | --- |
| מחלקה | Class |
| עצם | Object |
| הנוכחי | This |
| כותרת המחלקה | class header |
| שיטה, פעולה | Method |
| מופע | Instance |
| מאפיין, תכונה | Attribute |
| הרשאות גישה | access specifiers |
| פרטי | private |
| פומבי | public |
| הפניה | reference |
| העמסה | overloading |
| בנאי | constructor |
| בנאי ברירת מחדל | default constructor |
| בנאי מעתיק | copy constructor |
| ממשק | interface |
| מימוש | implementation |
| מנגנון התיעוד של JAVA | javadoc |
| טיפוס נתונים מופשט | abstract data type (ADT) |



### תרגיל לסיכום הפרק

נרחיב את המחלקה Point ונוסיף לה שתי פעולות:

|  |  |
| --- | --- |
| שאילתה המקבלת נקודה ומחזירה את המרחק שבינה לבין הנקודה הנוכחית (ראו למטה תזכורת לחישוב המרחק). | double distance (Point p) |
| שאילתה המקבלת נקודה ומחזירה את הנקודה הנמצאת בין הנקודה שהתקבלה כפרמטר ובין הנקודה הנוכחית באמצע. | Point middle (Point p) |

**תזכורת:** חישוב נקודת האמצע בין שתי הנקודות *(x1, y1)* ו-*(x2, y2)* הוא:

חישוב המרחק בין שתי נקודות*(x1, y1)* ו-*(x2, y2)* הוא:



פעולות החזקה והשורש קיימות במחלקה Math המופיעה במאגר המחלקות המוכנות של JAVA (ראו ב-(API. Math נמצאת בתוך החבילה הסטנדרטית java.lang, ולכן אין צורך לייבא אותה.

**ממשק חלקי של המחלקה Math**

המחלקה מאגדת בתוכה פעולות מחלקה שונות המבצעות חישובים מתמטיים נפוצים.

|  |  |
| --- | --- |
| שאילתה המקבלת שני פרמטרים x ו-y ומחזירה את הערך של x בחזקת y. | double pow (double x, double y) |
| שאילתה המקבלת את הפרמטר x ומחזירה את השורש הריבועי שלו. | double sqrt (double x) |

הפעולות האלה הן פעולות מחלקה. הפעלת הפעולות נעשית דרך שם המחלקה. לדוגמה, כדי להוציא שורש מ-9 יש לכתוב:

double root = Math.sqrt(9);

1. הוסיפו לפעולה הראשית חישוב של המרחק בין שתי הנקודות המקוריות שיצרתם בחלק א, והדפיסו אותו.
2. הוסיפו לפעולה הראשית חישוב של נקודת האמצע בין הנקודות שיצרתם בחלק א, לאחר החלפת ערכי ה-x. הדפיסו את הנקודה.

**שימו לב:** בפעולה המחשבת נקודת אמצע, ערך ההחזרה גם הוא עצם מסוג Point. כלומר עליכם ליצור את העצם החדש בתוך מימוש הפעולה, ואז להחזיר אותו כערך ההחזרה של הפעולה.

### תרגילים נוספים

משימה 1: תאריך (Date)

ממשק המחלקה Date

המחלקה מגדירה את הטיפוס תאריך, המורכב מיום, חודש ושנה.

|  |  |
| --- | --- |
| Date(int day, int month, int year) | בנאי היוצר מופע מטיפוס Date על פי פרמטרים נתונים.  **הנחות**: ערך הפרמטר day הוא מספר שלם בין 1 ל-31. ערך הפרמטר month הוא מספר שלם בין 1 ל-12. ערך הפרמטר year הוא מספר שלם אי-שלילי בן ארבע ספרות. |
| int getYear() | שאילתה המחזירה את השנה. |
| int getMonth() | שאילתה המחזירה את החודש. |
| int getDay() | שאילתה המחזירה את היום. |
| void setYear(int yearToSet) | פקודה קובעת את ערך השנה על פי הפרמטר הנתון.  **הנחה**: ערך הפרמטר הוא מספר שלם אי-שלילי בן ארבע ספרות. |
| void setMonth(int monthToSet) | פקודה הקובעת את ערך החודש על פי הפרמטר הנתון.  **הנחה**: ערך הפרמטר הוא מספר שלם בין 1 ל-12. |
| void setDay(int dayToSet) | פקודה הקובעת את ערך היום על פי הפרמטר הנתון.  **הנחה**: ערך הפרמטר הוא מספר שלם בין 1 ל-31. |
| int compareTo(Date other) | שאילתה המחזירה מספר חיובי אם התאריך הנוכחי מאוחר מהתאריך other; 0 – אם התאריכים שווים; מספר שלילי – אם התאריך הנוכחי קודם לתאריך other. |
| String toString() | שאילתה המחזירה מחרוזת המתארת את התאריך בצורה הבאה:  <day>.<month>.<year> |

**מה עליכם לעשות?**

**חלק א:**

1. כתבו את כותרת המחלקה Date, בחרו ייצוג למחלקה וממשו את כל הפעולות הנזכרות בממשק. ניתן להניח תקינות של כל הקלטים ואין צורך לבצע בדיקות תקינות לגביהם.
2. תעדו את המחלקה כראוי והפיקו קובץ API עבור המחלקה.
3. כתבו תוכנית בדיקה בשם TestDate, ובה בדקו את כל הפעולות שמימשתם במחלקה Date, כלומר צרו לפחות שני מופעים של Date שיפעילו את כל פעולות הממשק.

**חלק ב:**

לפניכם תוכנית ראשית המשתמשת במחלקה Date:

public static void main(String[] args) {

Date d1 = new Date(16, 7, 1963);

Date d2 = d1;

d1.setDay(20);

d2.setYear(1980);

System.out.println(d1);

System.out.println(d2);

}

1. מה יודפס בתום הרצת התוכנית?
2. כמה עצמים מסוג Date נוצרו במחלקה הראשית? הסבירו.

משימה 2: מספר רציונלי

רקע

מספר רציונלי הוא מספר שאפשר לייצגו כמנה של שני מספרים שלמים: מונה ומכנה. למשל, 0.3 הוא מספר רציונלי כיוון שאפשר לייצגו כך: .

טיפוס כזה כבר קיים ב-JAVA כטיפוס פשוט (double). עתה נגדיר אותו בתור מחלקה.

ניתן להגדיר מספר רציונלי בעזרת מחלקה בעלת שתי תכונות: מונה ומכנה, ששניהם מספרים שלמים.

הכפלת המונה והמכנה של מספר רציונלי באותו המספר מבטאת ייצוג אחר של אותו המספר.

לדוגמה: 

כלומר: יכולים להתקיים עצמים המייצגים את אותו מספר רציונלי, אף שערכי תכונותיהם שונים.

לא כל שני מספרים מייצגים מספר רציונלי חוקי. כאשר ערך התכונה המייצגת את המכנה הוא 0, המספר איננו חוקי.

ממשק המחלקה Rational

המחלקה Rational מגדירה מספר רציונלי.

תזכורת: בפעולות רבות יש להתחשב במקרי קצה בעייתיים. כאשר מדובר בקלט לפעולה, נעדיף להתריע בתיעוד הפעולה על הבעיה ולקבוע עבור אילו ערכים תפעל הפעולה כראוי. כך נמנע מהמשתמשים במחלקה להעביר ערכים לא רצויים לפעולה. בהמשך נלמד על מנגנון החריגות שמאפשר להתמודד עם מקרים אלה ולהציע להם פתרונות. נצטרך לבדוק מהן הבעיות העלולות להתעורר תוך כדי הפעולה ולפתור אותן במימוש הפעולה.

|  |  |
| --- | --- |
| בנאי המקבל שני פרמטרים: x עבור המונה ו-y עבור המכנה, ויוצר מופע של מספר רציונלי.  הנחה: המכנה אינו יכול להיות שווה 0, כלומר כל מספר שייווצר הוא מספר רציונלי חוקי. | Rational(int x, int y) |
| שאילתה המחזירה את המונה של המספר הנוכחי. | int getNumerator() |
| שאילתה המחזירה את המכנה של המספר הנוכחי. | int getDenom() |
| שאילתה המקבלת כפרמטר מספר רציונלי נוסף num, מחזירה אמת אם המספר הרציונלי הנוכחי ו-num שווים זה לזה, ושקר אחרת. | boolean isEqual(Rational num) \* |
| פקודה המקבלת כפרמטר מספר רציונלי נוסף num, ומחזירה מספר רציונלי שהוא מכפלת הפרמטר במספר הנוכחי. | Rational multiply(Rational num) |
| פקודה המקבלת כפרמטר מספר רציונלי נוסף num, ומחזירה מספר רציונלי שהוא המנה של המספר הנוכחי ב-num. יש לבדוק שהמחלק אינו בעל מונה שווה ל-0, אם המחלק אכן לא תקין תחזיר הפעולה null. | Rational divide(Rational num) \*\* |
| שאילתה המחזירה מחרוזת המתארת את המספר הרציונלי בצורה הבאה: <y>/ <x>. | String toString() |

\* מוכל להשוות בין מספרים רציונליים בעזרת מכפלת המונה של האחד במכנה של השני והשוואת המכפלות. לדוגמה:

\*\* החלוקה של שבר בשבר נעשית על ידי כפל השבר הראשון בהופכי של השבר השני. לדוגמה:



**מה עליכם לעשות?**

**חלק א**

1. כתבו את המחלקה Rational. הקפידו לתעד כראוי את הבנאי תוך ציון הדרישה שהמכנה לא יהיה שווה 0.
2. כתבו תוכנית בדיקה הבודקת את המימוש שכתבתם עבור פעולות הממשק.
3. הפיקו מסמך HTML המכיל את התיעוד המלא של המחלקה Rational.

**חלק ב**

קיבלתם את המחלקה Rational כמחלקה קיימת, ואינכם יכולים לשנותה. אתם מעוניינים בשתי פעולות נוספות: בפעולת חיבור ובפעולת חיסור של שני מספרים רציונליים.

1. איך תוכלו להגדיר ולממש פעולות אלה, והיכן?
2. כתבו את כותרות הפעולות המתאימות וכן תיעוד מלא לפעולות אלה.
3. ממשו את פעולת החיבור של שני מספרים רציונליים.
4. כתבו תוכנית בדיקה שתוכיח את נכונות הפעולות שמימשתם.



עבודת סיכום – שער ראשון

לפניכם עבודה מסכמת חשובה לכל מה שלמדתם בשער הראשון. לקראת המעבר לשעבר השני, חשוב שתבצעו את המשימה הזו במלואה. בהצלחה!

בבית הספר שלכם רוצים להקים מערכת מידע המטפלת בתלמידים. עבור כל תלמיד ותלמידה (Student) ייבנה עצם המכיל את פרטיהם: שם מלא ומספר טלפון.

לפניכם תרשים UML עבור המחלקהStudent .

ממשו את המחלקה בסביבת העבודה.

String name

String phoneNum

# Student

Student (String name, String phoneNum)

String getName()

String getPhoneNum()

void setPhoneNum (String phoneNum)

String toString()

צוות בית הספר מעוניין שהיישום של המערכת הבית-ספרית יאפשר חלוקה לקבוצות של תלמידים לפי שכבה, מחזור וכיתה. ניצור מחלקה בשם StudentList שתשמש כתבנית לייצור רשימות כיתתיות של תלמידים. לפניכם ממשק המחלקה המאפשר הכנסת תלמיד או תלמידה לרשימה כיתתית, הוצאתם מהרשימה, קבלת פרטיהם על פי השם, והדפסת דף קשר כיתתי ממוין בסדר א"ב. היישום כולו מורכב בשלב זה מהיחידות הבאות: המחלקה Student ומאגר תלמידים שיטופל במסגרת המחלקה StudentList.

ביישום המערכת אנו עוסקים בעולם מוקטן ומוגבל, שבו אדם מזוהה על פי שמו ולא ייתכן שיהיו קיימים שמות כפולים. כך תחסכו את העיסוק בהנחות בתיעוד ובבדיקות בתוך הקוד. כמו כן נניח שיש דרך פשוטה לדעת אם עוד יש מקום להכניס תלמידים נוספים לכיתה בלי להוסיף למחלקה תכונה ופעולות חדשות.

ממשק המחלקה StudentList

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

|  |  |
| --- | --- |
| בנאי היוצר רשימה כיתתית ריקה. | Rational(int x, int y) |
| פקודה המוסיפה את התלמיד/ה st לרשימה הכיתתית. | int getNumerator() |
| פקודה המוחקת את התלמיד/ה בשם name מתוך הרשימה הכיתתית. הפעולה מחזירה את התלמיד/ה שנמחקו. אם התלמיד/ה אינם קיימים ברשימה, הפעולה מחזירה null. | int getDenom() |
| שאילתה המחזירה את התלמיד/ה בשם name. אם התלמיד/ה אינם קיימים ברשימה, הפעולה מחזירה null. | boolean isEqual(Rational num) \* |
| שאילתה המחזירה מחרוזת של דף קשר כיתתי הממוין לפי א"ב באופן זה:  <name1> <tel1>  <name2> <tel2> | Rational multiply(Rational num) |

הנחיות

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

שלבים בכתיבת המחלקה StudentList:

א . ייצוג המחלקה

פותרים ביחד – מה צריכות להיות תכונות המחלקה? (במילים אחרות, יצגו את המחלקה).

תשובה מוסתרת

כיון שעצם של המחלקה מחזיק מידע על מספר רב של תלמידים, לא נכון להגדיר תכונה עבור כל תלמיד ותלמידה. סביר יותר להגדיר תכונה מטיפוס מערך שתכיל את אוסף התלמידים. מספר התלמידים הממוצע בכיתה בבית הספר הוא 30. לפי כללי בית הספר אין אפשרות שמספר התלמידים יעלה על 35, לכן קובעים שגודל המערך יהיה תמיד 35.

להלן ייצוג המחלקה וקטע התוכנית המממש חלק מהפעולות:

public class StudentList {

public static final int MAX\_STUDENTS = 35;

private Student[] list;…

}

ממשו את הבנאי של המחלקה

פתרון :

public StudentList (){

this.list = new Student[MAX\_STUDENTS];

}

**חושבים ביחד**

מה ערכו של כל איבר במערך לפני שהכנסנו לתוכו תלמידים?

במסגרת הבנאי של המחלקה StudentList מוגדר מערך ללא אִתחול. כיוון שזהו מערך של הפניות, המערכת מאתחלת כל תא במערך להפניה הריקה null. כדי להתחיל שנת לימודים תקינה יש להכניס למערך את תלמידי הכיתה.

פעולת ההוספה

הפעולה האחראית על הכנסת התלמידים למערך היא פעולת ההוספה, והיא אמורה להכניס תלמיד או תלמידה לתא הראשון הפנוי במערך בכל שלב.

**חושבים ביחד**

כיצד יודעים מהו המקום הראשון הפנוי במערך?

תשובה

ניתן לסרוק את המערך בעת ביצוע פעולת הכנסה כדי להגיע לתא הראשון שערכו null. זהו המקום המתאים להכנסה הנוכחית.

**חושבים ביחד**

נסו לחשוב על פתרון אחר (רמז – שימוש במשתנה / תכונה)

מימוש יעיל יותר של פעולה זו אפשרי אם מגדירים תכונה נוספת, lastPosition, במחלקה StudentList. תכונה זו תשמור את האינדקס של המקום הפנוי הראשון. אינדקס זה יתעדכן בזמן ההוספה של תלמידים חדשים לרשימה הכיתתית. במימוש הנוכחי עליכם לפעול על פי גישה זו. בבנאי אתחלו את התכונה לאפס, ובפעולת ההוספה הכניסו את התלמיד החדש למקום lastPosition:

המחלקה כפי שמימשנו אותה עד כה

public class StudentList {

public static final int MAX\_STUDENTS = 35;

private Student[] list;

private int lastPosition;

public StudentList() {

this.list = new Student[MAX\_STUDENTS];

this.lastPosition = 0;

}

public void add(Student st) {

this.list[this.lastPosition] = st;

this.lastPosition++;

}

שאלה למחשבה:

בעת זימון פעולת ההוספה מועבר אליה פרמטר שהוא הפניה לתלמיד החדש.

היכן נוצר התלמיד החדש?

כמה הפניות יש אל התלמיד החדש בסיום פעולת ההוספה?

StudentList lst1 = new StudentList( ) ;

מוצג תרשים עצמים הכולל בתוכו הפניה למערך . במערך 35 איברים מ-0 ועד 34, שכל אחד מהם הוא null.

Student student1 = new Student ("Tamar" , "0508989898");

Student student2 = new Student ("Ran" , "0527171717");

מוצג תרשים עצמים עבור כל אחד מהתלמידים.

lst.add(student1) ;

מציגים שוב את העצם עם המערך, והפעם יש הפניה מהמציין 0 במערך אל העצם student1.

גם בתרגיל הביניים הקטן הזה בוודאי הבחנתם כי לעצם אחד יכולות להיות הפניות שונות. פעולות עדכון ושינוי של העצם דרך הפניה אחת תשתקף גם כשפונים אליו בעזרת ההפניות האחרות, ועלינו להיות מודעים לכך.

פעולת המחיקה:

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

לפניכם קטע התוכנית לפעולה del(…):

public Student del(String name) {

Student st = null;

int i = 0;

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

i++;

if (i < this.lastPosition) {

st = this.list[i]; // צמצום "חורים"

for (int k = i + 1; k < this.lastPosition; k++)

this.list[k - 1] = this.list[k];

this.lastPosition--;

this.list[this.lastPosition] = null;

}

return st;

}

C#

public Student del(String name) {

Student st = null;

int i = 0;

while (i < this.lastPosition && this.list[i].GetName().CompareTo(name) != 0)

i++;

if (i < this.lastPosition) {

st = this.list[i]; // צמצום "חורים"

for (int k = i + 1; k < this.lastPosition; k++)

this.list[k - 1] = this.list[k];

this.lastPosition--;

this.list[this.lastPosition] = null;

}

return st;

}

הפעולה toString():

**רגע חושבים!**

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

האם נוכל להימנע ממיון הרשימה בביצוע הפעולה toString()?

אם נוודא שהמערך המייצג את הרשימה הכיתתית נשמר ממוין כל הזמן, לא נצטרך להשקיע משאבים במיונו לקראת הפקת דף הקשר. ואולם לשם כך יהיה עלינו לממש אחרת את פעולות ההוספה והמחיקה, כך ששם כל תלמיד מוכנס אל המערך או מוצא ממנו על פי מקומו הנכון בסדר הא"ב.

בסעיף להלן נפגוש מימוש חילופי שכזה.

ו.3 מימוש חילופי

האם הייצוג של המחלקה משתנה בעקבות ההחלטה על שינוי המימוש? לכאורה ניתן היה לוותר על הגדרת התכונה lastPosition. בכל זאת, עדיף להשאיר את התכונה ולהיעזר בה, ועל כן להשאיר את הייצוג כפי שהיה, ולשנות רק את מימוש הפעולות הבאות.

פעולת ההוספה

כתיבה מחודשת של פעולת ההוספה, כך שהמערך יישמר ממוין כל הזמן, תתבצע כך: המערך הראשוני הנוצר עבור הרשימה אינו מכיל תלמידים, ולכן הוא ממוין. כעת יש לוודא שכל תלמיד שיוכנס אליו, יוכנס ישירות למקומו הנכון על פי סדר הא"ב. כך יישמר המערך ממוין כל זמן.

האלגוריתם של פעולת ההוספה השומרת על מיון המערך ייראה כך:

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

בשלב הראשון של האלגוריתם נמצא את המקום שאליו צריך להכניס את התלמיד או התלמידה החדשים. זהו המקום שבו הסריקה נעצרת. הנחת המוצא של היישום הבית-ספרי הייתה שאין שמות כפולים כלל; לכן תנאי הסריקה הוא שבנקודת העצירה שמות התלמידים הקודמים קטנים משם התלמיד החדש, והשם במקום הנוכחי כבר גדול ממנו. לכן המקום שהתפנה הוא המקום שאליו יש להכניס את התלמיד/ה. כדי לשמור על הסדר יש

להזיז את כל התלמידים ממקום זה והלאה מקום אחד ימינה. ההנחה המקדימה הנוספת ביישום אומרת שתמיד יהיה מקום במערך להכנסת תלמידים חדשים, ואין צורך לבדוק זאת.

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

הקוד שלפניכם מממש את האלגוריתם המתואר תוך שימוש בפעולת העזר moveOthers(…) המממשת את שלב 2 באלגוריתם. במקרה הקצה, כשהוספת התלמיד או התלמידה נעשית בסוף המערך, אין צורך להפעיל את הפעולה:

public void add(Student student) {

int i = 0;

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

i++;

if (i < this.lastPosition)

moveOthers(i);

this.list[i] = student;

this.lastPosition++;

}

C#

public void add(Student student) {

int i = 0;

while (i < this.lastPosition && this.list[i].GetName().CompareTo(student.GetName()) < 0)

i++;

if (i < this.lastPosition)

MoveOthers(i);

this.list[i] = student;

this.lastPosition++;

}

ממשו את הפעולה void moveOthers (int place).

לפניכם מחלקה בשם TestSchool. המחלקה יוצרת רשימה כיתתית ובודקת את תקינות הפעולות שכתבנו במחלקה StudentList . הוסיפו אותה לפרויקט והריצו אותה.

הערות לסיכום

בעבודה זו התמודדנו לראשונה עם יישום "כמעט אמיתי" של מערכת לניהול בית-ספרי. המערכת משתמשת בכמה נתונים – עצמים מהטיפוס Student ואוספים של עצמים אלה בצורת רשימות כיתתיות מטיפוס StudentList.

הטיפוס Student מגדיר עצמים פשוטים, אך האוספים הם עצמים מורכבים המכילים הפניות לעצמים רבים אחרים. לא נוכל לדון כאן בהרחבה כיצד נכון לארגן יישום מורכב כזה, ולכן נסתפק בשתי הערות:

1. ברשימה כיתתית משתמשים בשם כמזהה של התלמידים, ומניחים שאין שני תלמידים בעלי אותו השם. ביישום אמיתי הנחה זו אינה תקפה, כיוון שייתכנו שני תלמידים ויותר בעלי אותו השם. לכן מקובל להשתמש במספר תעודת הזהות כמזהה. השתמשנו כאן בשמות לשם הפשטות והנוחות בלבד.

עד עכשיו כאשר הגדרנו מחלקות חדשות, התכונות שבהן היו בדרך כלל פרטיות ולכל תכונה הגדרנו פעולות קריאה ושינוי – get(), set(…). אולם אין חובה לספק פעולות אלה לכל תכונה.

2. כבר הזכרנו כי ייתכן שיהיו כמה הפניות לאותו העצם. גם ביישום של הרשימות הכיתתיות יש הפניות לתלמידים. כפי שלמדנו, אפשר לשנות עצם מכל הפניה אליו, אך אם נרצה להשתמש ברשימה הכיתתית לצרכים שונים אסור לעשות זאת דרך הפניה לרשימה עצמה. דרך השימוש הנכונה היא ביצירת עותק של הרשימה בעזרת בנאי מעתיק.