

פרק 2 עצמים שימוש במחלקות

עד היום הכרתם טיפוס ערכים פשוטים המוגדרים בשפה כמו `int` ו-`double`, יצרתם משתנים היכולים להכיל ערכים מטיפוסים אלה והשתמשתם בהם לפתרון בעיות. על ערכים פשוטים אלה אפשר היה לבצע פעולות שונות: חיבור, חיסור, מציאת יחס הסדר בין הערכים ועוד. הכרתם גם טיפוס ערכים שאינם פשוטים, הכלולים בשפה, והכרתם את הפעולות הייחודיות להם. אחד מהטיפוסים הללו הוא המחרוזת, `String`, המאפשרת לטפל באוסף של תווים המאוגדים יחד ולבצע עליהם מגוון רחב של פעולות: שרשור, החזרת תת-מחרוזת, החלפת תווים או תת-מחרוזות בתוך המחרוזת המקורית ועוד. טיפוס נוסף מסוג זה הוא המערך, `array`, המאפשר לטפל באוסף של נתונים, לברר את אורכו, וכן לגשת ישירות לכל אחד מאיברי האוסף, לשלוף אותו, לעדכן אותו ועוד.

א. טיפוסים חדשים

כדי לפתח תוכנות מעניינות, נרצה להגדיר בעצמנו טיפוסים חדשים נוספים. עבור כל טיפוס כזה נגדיר הן קבוצת ערכים מסוג מסוים והן אוסף של פעולות שניתן לבצע על ערכים אלה. דוגמאות לטיפוסים חדשים כאלה הן: קוביות משחק, גופים הנדסיים שונים, תאריכים, חשבונות בנק ועוד. הטיפוסים החדשים יאופיינו על ידי הערכים שלהם והפעולות שניתן לבצע עליהם. בג'אווה, טיפוס חדש מסוג זה מוגדר על ידי **מחלקה** (`Class`). שימו לב: כפי שמשתנים מטיפוסים פשוטים שיצרנו הכילו ערכים מסוג הטיפוס, כך גם משתנים מטיפוס מחלקה כלשהי יכילו ערכים מטיפוס המחלקה. ערכים אלה ייקראו עצמים. לכל **עצם** (`Object`) נקבע **מצב** (`state`) משל עצמו בזמן יצירתו. כל עצם יכול לבצע פעולות המדווחות על מצבו או משנות אותו. מחלקה כוללת הגדרת הייצוג של מצבי העצמים, קוד של הפעולות וכן מנגנון המאפשר לייצר עצמים במצב תחילי רצוי. בתוכנית כלולות בדרך כלל כמה מחלקות. התוכנית מייצרת מהן עצמים, שעל ידי הפעלתם מתבצעת המשימה שלשמה נכתבה התוכנית. על כתיבת מחלקות נלמד בפרק הבא. בפרק זה ניצור עצמים ממחלקות נתונות, ונשתמש בהם למילוי משימות, כלומר נלמד ונתרגל את אופני הייצור של עצמים ואת השימוש בהם.

ב. מצב של עצם

כאמור, לכל עצם יש מצב. כשמבוצעת הוראה ליצירת עצם, נוצר עצם חדש ונקבע מצבו. כך למשל: לאחר שנכתבה המחלקה `Box`, קיימת הגדרה של טיפוס בשם זה. כל עצם הנוצר ממחלקה זו הוא מטיפוס `Box`.

```
Box b;
```

הפקודה:

מגדירה משתנה מטיפוס `Box` שיוכל להכיל עצם כלשהו מטיפוס זה.

הפקודה:

```
b = new Box(10, 12, 20);
```

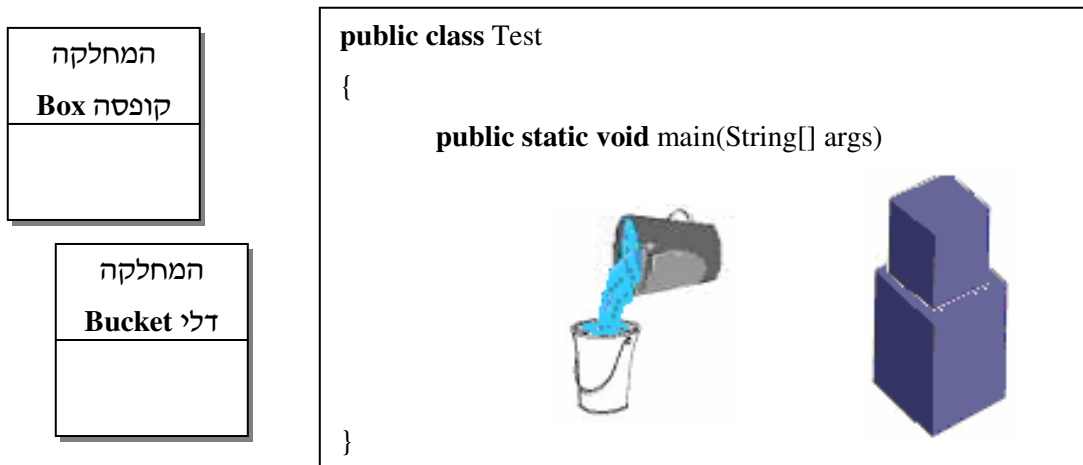
יוצרת עצם חדש מטיפוס זה ומכניסה אותו למשתנה b.

מצבו של עצם מטיפוס "קופסה" ניתן לאפיון על ידי אורך, רוחב וגובה. מצב הקופסה נקבע בזמן יצירתה. כך בדוגמה שלמעלה, האורך, הרוחב והגובה הם הערכים 10, 12 ו-20 בהתאמה. עצמים שונים מאותו טיפוס יכולים כמובן להיות בעלי מצבים שונים, למשל לקופסאות שונות יש ממדים שונים. כמו כן אפשר שלעצמים שונים שנוצרו מאותה מחלקה יהיו מצבים זהים, כשם שייתכן שלקופסאות שונות יהיו אותם ממדים ובכל זאת הם יהיו מופעים שונים של אותה מחלקה. מצבם של העצמים יכול להשתנות במהלך תוכנית.

הפעולות (methods) שעצם מבצע, מכירות את מצב העצם ויכולות לאחזר ערכים המתארים את מצב העצם או לשנותם. למשל, ניתן להגדיר עבור הטיפוס "קופסה" את הפעולות הבאות: אחזור גובה הקופסה, שינוי הממדים של הקופסה, חישוב נפח הקופסה וחישוב שטח הפנים שלה. כאשר עצם מטיפוס "קופסה" מבצע, למשל, את פעולת אחזור הגובה, מצבו של העצם מוכר לפעולה, ולכן היא יכולה למצוא את ערך גובהו ולהחזירו. העצם מבצע פעולה כאשר הוא מקבל הודעה מתאימה.

ג. תוכנית

באופן כללי, תוכנית מכילה הגדרות של מחלקות. במהלך ביצועה נוצרים עצמים מהמחלקות. עצמים אלה מבצעים פעולות כאשר נשלחות אליהם הודעות. האיור הבא מדגים מבנה של תוכנית. התוכנית משתמשת בעצמים משני טיפוסים: "קופסה" ו"דלי". לצורך הגדרת טיפוסים אלה נכתבו שתי מחלקות. בנוסף, נכתבה מחלקה שלישית, Test, ובה מופיעה הפעולה הראשית, שבה מתחיל ביצוע התוכנית. בפעולה הראשית נוצרים עצמים מטיפוסי שתי המחלקות, ואחר כך נשלחות אל עצמים אלה הוראות לביצוע פעולות שונות. שני העצמים יכולים להעביר הודעות ביניהם. ביצוע הפעולות הנדרשות על ידי ההודעות מביא למילוי המשימה של התוכנית.



באיור זה Test היא המחלקה הראשית. היא מכילה את קטע הקוד היוצר עצמים מהמחלקות קופסה ודלי ומעביר אל העצמים הודעות לביצוע פעולות שונות.

לתהליך של תכנות מונחה עצמים שני שלבים :

1. כתיבת מחלקות המגדירות טיפוסים חדשים.
2. כתיבת פעולה ראשית היוצרת מהמחלקות עצמים ומפעילה אותם.

בפרק זה נלמד כיצד ליצור עצמים ממחלקות שכבר הוגדרו ולהשתמש בהם. משימות אלה יבוצעו בעזרת מחלקה המכילה את הפעולה הראשית. נזכיר כי תוכנית בג'אווה נכתבת בקובץ הנושא את שם התוכנית (מתחיל באות גדולה בלבד) ומסתיים בסיומת: '.java'. על מוסכמה זו ועל מוסכמות נוספות נפרט בפרק הבא, ובו נלמד גם לכתוב מחלקות.

ד. ממשק

לכל מחלקה מצורף מסמך המפרט כיצד להשתמש בה. למסמך זה קוראים **ממשק (interface)**. בממשק מפורטות הפעולות שעצם יכול לבצע וכן פעולות בונות, המגדירות דרכים לבניית עצמים. עבור כל פעולה מופיעה **הכותרת (header)** שלה ובה: שם הפעולה, טיפוסים הפרמטרים שהיא מקבלת וכן טיפוס הערך שהיא מחזירה. הממשק מכיל גם תיאור מילולי של הפעולה. בדרך כלל מכיל הממשק תיאור של פעולה בונה אחת או יותר. ההסכם המחייב בשפת ג'אווה הוא ששם פעולה בונה 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. עיקרון זה נקרא: **הפרדה בין ממשק למימוש**. כפי שנלמד בהמשך, קיים מנגנון פשוט המייצר תיאור של ממשק המחלקה מהתיעוד הפנימי שלה.

המושגים הללו: "ממשק", "מימוש", "תיעוד" ו-"ההפרדה בין ממשק למימוש" אינם ייחודיים לתכנות, הם משמשים אותנו בטבעיות בתחומים רבים. לדוגמה כאשר אנו קונים טלפון סלולרי חדש, אין צורך שנדע מהו המבנה הפנימי שלו. מספיק שנקבל חוברת הדרכה המפרטת כיצד מפעילים את כל האפשרויות שהמכשיר מציע.

דוגמה נוספת: כאשר אנו משתמשים בתוכנה מסוימת שנכתבה עבורנו, למשל Word, אין צורך לדעת איך היא נכתבה וכיצד היא ממומשת, מספיק שנדע היכן להקליק עם העכבר ובאילו תפריטים לחפש את מה שנחוץ לנו.

לפניכם הממשק של המחלקה "קופסה" – Box. העמודה הימנית מציגה את תיאור הפעולה ואילו העמודה 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. פעולה בונה אינה פעולה של עצם כלשהו, והיא מופעלת על ידי המחלקה. גם עבורה אין צורך בעצם כפרמטר.

ממשק המחלקה Box

המחלקה Box מגדירה קופסה תלת-ממדית לפי הערכים אורך, רוחב וגובה.

Box (double length, double width, double height)	פעולה בונה של המחלקה קופסה. הפעולה מקבלת 3 פרמטרים: אורך, רוחב וגובה ומאתחלת את מצב הקופסה החדשה לפי ערכי הפרמטרים. הנחה : הפרמטרים הם מספרים אי-שליליים
double getLength()	הפעולה מחזירה את האורך של הקופסה
double getWidth()	הפעולה מחזירה את הרוחב של הקופסה
double getHeight()	הפעולה מחזירה את הגובה של הקופסה
void setLength (double length)	הפעולה משנה את האורך לפי הערך המתקבל כפרמטר. הנחה : הפרמטר הוא מספר אי-שלילי
void setWidth (double width)	הפעולה משנה את הרוחב לפי הערך המתקבל כפרמטר. הנחה : הפרמטר הוא מספר אי-שלילי
void setHeight (double height)	הפעולה משנה את הגובה לפי הערך המתקבל כפרמטר. הנחה : הפרמטר הוא מספר אי-שלילי
double getVolume()	הפעולה מחשבת ומחזירה את הנפח של הקופסה
String toString()	הפעולה מחזירה מחרוזת המתארת את הקופסה

ה. פעולה בונה

פעולה בונה (constructor) מאפשרת לקבוע את מצבו של עצם חדש. שמה של פעולה זו הוא כשם המחלקה. שימו לב: פעולה בונה אינה מופעלת על ידי עצם מסוים אלא מתוך המחלקה. כך מתוארת הפעולה הבונה בממשק של המחלקה "קופסה" – Box:

Box (double length, double width, double height)	פעולה בונה של המחלקה קופסה. הפעולה מקבלת 3 פרמטרים: אורך, רוחב וגובה ומאתחלת את מצב הקופסה החדשה לפי ערכי הפרמטרים
--	--

זימון הפעולה הבונה מתבצע בעזרת המילה השמורה **new**, האחראית על יצירת עצם חדש מטיפוס המחלקה הנזכרת. למשל, כדי ליצור קופסה שאורכה 5, רוחבה 3.2 וגובהה 10, נכתוב:

```
new Box(5, 3.2, 10);
```

הוראה זו מייצרת עצם חדש שמצבו הוא מצב תחילי הנקבע על ידי המערכת. מיד אחר כך מופעלת הפעולה הבונה על עצם זה וקובעת את מצבו בהתאם לערכי הפרמטרים. כדי ליצור קופסה שאורכה 3, רוחבה 2 וגובהה 1.3, נכתוב:

```
new Box(3, 2, 1.3);
```

לעצם שיצרנו ממחלקה מקובל לקרוא **מופע (instance)** של המחלקה. בשלב זה של לימודינו, יצירת עצמים תעשה בדרך כלל בתוך הפעולה הראשית האחראית על מבנה התוכנית. בהמשך נראה כי הגדרת מחלקה יכולה להכיל כמה פעולות בונות, המגדירות דרכים שונות לקביעת מצבם ההתחלתי של עצמים הנוצרים מהמחלקה.

1. הפניות

עד כה הכרנו משתנים היכולים להכיל ערכים פשוטים, כגון מספרים, אבל משתנה יכול גם "להכיל" עצם. בפועל, המשתנה מכיל **הפניה (reference)** אל העצם. כאשר מוגדר משתנה מטיפוס מחלקה מסוימת אזי הוא יכול להכיל הפניות לעצמים שהם מופעים של המחלקה, כלומר עצמים מטיפוס המחלקה.

לדוגמה, ההגדרה הבאה המופיעה בפעולה הראשית:

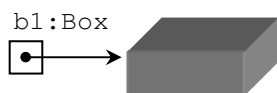
```
Box b1;
```

מגדירה משתנה b1 מטיפוס Box. בשלב זה אין ל-b1 ערך ועלינו לאתחל אותו.

ניצור מופע של Box ונציב את ההפניה אליו בתוך המשתנה:

```
b1 = new Box(5, 3, 2);
```

עכשיו b1 מכיל הפניה לעצם מטיפוס Box:



אפשר לאחד את הגדרת המשתנה ואת הצבת ההפניה לעצם החדש שנוצר במשתנה זה, ולכתוב:

```
Box b1 = new Box(5, 3, 2);
```

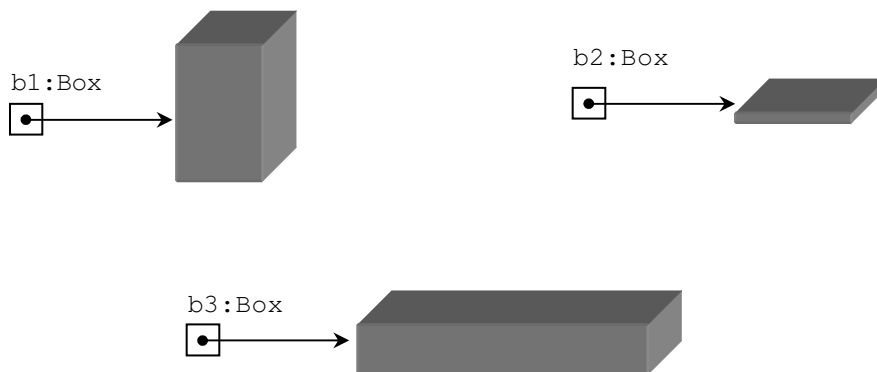
ההכרזה באגף שמאל מגדירה משתנה מטיפוס Box.

באגף ימין נוצר עצם מטיפוס Box שמצבו נקבע לפי ערכי הפרמטרים. באגף זה Box הוא שם הפעולה הבונה. ביצוע הפקודה מגדיר אם כן משתנה ועצם במצב מסוים, ומציב במשתנה את ההפניה לעצם.

עתה ניצור בפעולה הראשית כמה מופעים של Box:

```
public class Test
{
    public static void main(String[] args)
    {
        Box b1 = new Box(2, 4, 7.1);
        Box b2 = new Box(5, 3, 0.75);
        Box b3 = new Box(10, 2.5, 2);
    }
}
```

יצרנו שלוש קופסאות שונות שאליהן מפנים שלושה משתנים שונים מטיפוס Box:



ז. פעולות נוספות

אחרי שהעצם נוצר הוא יכול לבצע את הפעולות המוגדרות בממשק (פרט לפעולות בונות). פעולות יכולות לקבל ערכים כפרמטרים וכן להחזיר ערכים. ערכים אלה יכולים להיות מטיפוסים פשוטים או מטיפוסי מחלקות.

כאשר רוצים לברר פרטים על מצבה של קופסה, משתמשים בפעולות המחזירות את הפרטים המבוקשים. אלה הן הפעולות `getLength()`, `getWidth()`, `getHeight()` שבממשק המחלקה `Box`. פעולות אלה של הקופסה אינן מקבלות פרמטרים, אך למרות זאת חובה לכתוב סוגריים מיד לאחר שם הפעולה. לשלוש הפעולות האלה יש ערך החזרה מטיפוס `double`.

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 המפנה אל העצם, מימין להפניה נכתוב נקודה, מימינה את שם הפעולה המבוקשת ואחריה את ערכי הפרמטרים בסוגריים:

```
Box b1 = new Box(2, 2, 2);
double x = b1.getLength();
```

בשורה הראשונה יצרנו עצם מטיפוס `Box`, קופסה מרובעת שאורכה, רוחבה וגובהה הם 2. את ההפניה לעצם הצבנו בתוך משתנה מטיפוס `Box` הנקרא `b1`. בשורה השנייה פנינו ל-`b1` בשיטת

סימון-נקודה כדי להפעיל את הפעולה `getLength()`. פעולה זו מחזירה את אורך הקופסה, שאותו אנו מציבים במשתנה מטיפוס `double`.

נזכיר כי פעולה בונה מזומנת באופן שונה, באמצעות המילה השמורה `new`. זו אינה פעולה המתבצעת על ידי עצם, אלא על ידי המחלקה.

נדגים זימון של פעולה המחזירה ערך. הזימון מופיע בתוך הפעולה הראשית:

```
public static void main(String[] args)
{
    Box b1 = new Box(2, 4, 6);
    double x = b1.getWidth();
    System.out.println(x);
}
```

בשורה הראשונה של הפעולה הראשית יצרנו מופע של `Box` והצבנו הפניה אליו בתוך משתנה מטיפוס `Box` הנקרא `b1`. בשורה השנייה הפעלנו את `getWidth()` באמצעות ההפניה `b1`. פעולה זו מחזירה את הרוחב של הקופסה. הערך המוחזר מוצב לתוך המשתנה `x` שהוא מטיפוס `double`. כיוון שהקופסה נוצרה ברוחב 4, ורוחבה לא שונה בתוכנית, הערך של `x` הוא 4, והפעולה הראשית תדפיס בשורה האחרונה את הערך הזה.

קיים סוג נוסף של פעולות והוא פעולות המשנות את מצבו של העצם ואינן מחזירות ערך. כדי לציין שטיפוס ערך החזרה שלהן הוא ריק נכתוב `void`. הפעולה `setLength(...)` מקבלת כפרמטר ערך מטיפוס `double` ששמו `length`, ומשנה את אורך הקופסה לערך החדש שמתקבל כפרמטר.

בממשק של הקופסה יש שתי פעולות נוספות: `setWidth(...)`, `setHeight(...)`, המשנות את רוחב הקופסה ואת גובהה. כמו כן כלולה בממשק פעולה נוספת, `getVolume()`, שאינה מקבלת פרמטרים. הפעולה מחשבת את נפח הקופסה ומחזירה ערך מטיפוס `double` המכיל אותו.

זימון הפעולה:

```
public static void main(String[] args)
{
    Box b1 = new Box(2, 4, 6);
    Box b2 = new Box(1, 1, 1);
    double x1 = b1.getVolume();
    double x2 = b2.getVolume();
    System.out.println(x1);
    System.out.println(x2);
}
```

? מה יודפס כתוצאה מהרצת הפעולה הראשית?

הפעולה האחרונה בממשק המחלקה קופסה, `toString()`, מחזירה מחרוזת המתארת את הקופסה כפי שהמתכנת ניסח.

המחרוזת יכולה להיות :

"The length of the box is 2, the width of the box is 4, the height of the box is 6"

"A box: 2, 4, 6"

אך היא יכולה להיות גם קצרה בהרבה :

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)
{
    Box b1 = new Box(2, 4, 6);
    String str = b1.toString();
    System.out.println(str);
}
```

בשורה הראשונה בפעולה הראשית נוצר מופע של Box שאורכו 2, רוחבו 4 וגובהו 6. עצם זה מוצג בהפניה b1. בשורה השנייה מופעלת הפעולה toString() שמחזירה מחרוזת המתארת את הקופסה (בהתאם לממשק). מחרוזת זו מוצבת בהפניה מטיפוס מחרוזת, הנקראת str. בשורה האחרונה המחרוזת הזו מודפסת.

שימו לב, ניתן להשתמש בכתוב מקוצר באופן הבא :

```
public static void main(String[] args)
{
    Box b1 = new Box(2, 4, 6);
    System.out.println(b1);
}
```

כאשר מתבקשת הדפסה של אובייקט, השפה מחפשת את פעולת ה- toString() המוגדרת לגבי אותו עצם ומפעילה אותה, ואין צורך לכתוב במפורש :

```
System.out.println(b1.toString())
```

ח. אתחול משתנים

משתנים מקומיים, הן מטיפוס בסיסי והן מטיפוס מחלקה, המוגדרים בתוכנית הראשית או בתוך פעולה, אינם מקבלים ערך ראשוני ידוע כלשהו ולכן אנו חייבים לאתחל אותם לערך הרצוי. אם לא נאתחל את המשתנים המקומיים לפני השימוש בהם, נקבל הודעה על שגיאת הידור והתוכנית לא תעבור קומפילציה.

לדוגמה :

```
Box b1;
b1.setWidth(5);
```

הפקודה הראשונה מגדירה משתנה b1, שעדיין אין לו ערך. הפקודה השנייה היא הוראה לבצע את הפעולה setWidth(...) של העצם שהפניה אליו מאוחסנת במשתנה. כיוון שהעצם כלל לא נוצר, הקומפילציה תיעצר ואנו נקבל הודעה על שגיאת הידור.

ט. מחלקות מוכנות

לשפת ג'אווה יש אוסף עשיר של ספריות המכילות מחלקות מוכנות לשימוש. השימוש במחלקות אלה נעשה בעזרת הממשק המתועד שלהן. אוסף הממשקים של מחלקות אלה נקרא: **JavaAPI** (Java Application Programming Interface), ואת תכולתו ניתן לראות מכל סביבת עבודה. ג'אווה מחזיקה את המחלקות הללו מסודרות בחבילות. **חבילה** (**package**) היא אוסף של מחלקות העוסקות באותו נושא. כדי להשתמש במחלקה עלינו "לייבא" אותה לתוכנית שבה אנו זקוקים לה. ה"ייבוא" נעשה באמצעות המילה השמורה **import**, והוא יכול להיות של כל החבילה המכילה את המחלקה הנדרשת או של המחלקה בלבד.

לדוגמה, כאשר רוצים להשתמש במחלקה **Point** (הסבר על המחלקה ראו בתרגילים), יש לבדוק באיזו חבילה היא נמצאת. שם החבילה מצוין בתחילת דף ה-**JavaAPI** של המחלקה, מעל שם המחלקה. המחלקה **Point** נמצאת בחבילה **java.awt**. ניתן לייבא את החבילה בשלמותה לתוך התוכנית שלנו על ידי כתיבת השורה הבאה:

```
import java.awt.*;
```

השורה צריכה להופיע בראש הקובץ, לפני כותרת המחלקה. הסימן כוכבית מצביע על "ייבוא" של כל המחלקות השמורות באותה חבילה. כדי לייבא רק את המחלקה הנדרשת יש לכתוב את השורה הבאה בראש הקובץ:

```
import java.awt.Point;
```

הערה חשובה: לצורך העבודה והתרגול של יחידה זו הכנו עבורכם מחלקות שונות, בעיקר מחלקות גרפיות. כדי להקל על עבודתכם, אורגנו המחלקות בספריות, והספריות נאספו לספרייה יחידה הנקראת **unit4**. ספרייה זו תיטען לתוך סביבת העבודה אוטומטית עם התקנת סביבת העבודה. בראש כל תוכנית המשתמשת במחלקה מאוסף זה יש לכתוב את השורה הבאה:

```
import unit4.<שם המחלקה>.<שם החבילה>;
```

אף שהמחלקות ממקור שונה, תוכלו לפנות לתיעוד של כל מחלקה באופן ישיר מתוך סביבת העבודה. אזכור שם המחלקה הוא בגדר רשות וניתן לוותר עליו ולציין:

```
import unit4.<שם החבילה>.*;
```

במקרה זה נוכל להשתמש בכל אחת מהמחלקות שבחבילה זו.

ט.1 המחלקה **String**

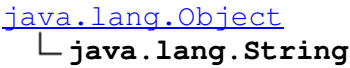
יש מחלקות שאין צורך "לייבא" אותן באופן מפורש, מכיוון שהן "מיובאות" על ידי סביבת העבודה באופן אוטומטי. מחלקות אלה נמצאות בחבילה **java.lang**. בחבילה זו נמצאות מחלקות שמרבים מאוד להשתמש בהן, ולכן הליך ההבאה שלהן קוצר. בחבילה זו שמורה גם המחלקה **String**. משום שהשימוש במחלקה זו רב, קוצרה גם דרך היצירה של מופעים מסוגה: וכך אין צורך להשתמש במילה השמורה **new** כדי ליצור עצם.

String str = "shalom"; : בוודאי מוכרת לכם הצורה הבאה ליצירת המחרוזת:

זוהי כתיבה מקוצרת הגורמת להצבת עצם מטיפוס מחרוזת שתוכנו הוא המחרוזת "shalom", בהפניה str. אם כבר קיימת בתוכנית מחרוזת שזה תוכנה, ולא משנה איזו הפניה מפנה אליה, לא ייווצר מופע חדש, נוסף, בעל אותו ערך. במקום זאת, תיווצר רק הפניה חדשה שתפנה אל אותו המופע הקיים. למעשה אל מופע זה יהיו מעכשיו שתי הפניות. נושא זה יורחב בהמשך היחידה.

יחודים נוספים של המחלקה String ועצמים מטיפוס זה יפורטו בתרגול של פרק זה ובהמשך היחידה. בתרגול של פרק זה מופיע גם חלק מממשק המחלקה String. לפניכם קטע מה-JavaAPI של המחלקה "מחרוזת".

java.lang
Class String



The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. ...

Constructor Summary	
<p>String()</p> <p>Initializes a newly created String object so that it represents an empty character sequence.</p>	
<p>String(String original)</p> <p>Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.</p>	
Method Summary	
char	<p>charAt(int index)</p> <p>Returns the char value at the specified index.</p>
int	<p>compareTo(String anotherString)</p> <p>Compares two strings lexicographically.</p>
int	<p>compareToIgnoreCase(String str)</p> <p>Compares two strings lexicographically, ignoring case differences.</p>
String	<p>concat(String str)</p> <p>Concatenates the specified string to the end of this string.</p>
boolean	<p>endsWith(String suffix)</p> <p>Tests if this string ends with the specified suffix.</p>

ביחידה זו אנו מפשטים את אופן הכתיבה של הממשק כפי שיובא בטבלה להלן. לפניכם דוגמה מתוך ממשק המחלקה `string`, ובה מוצג רק מיעוט פעולות הממשק. תיעוד הפעולות מובא באופן חלקי:

<code>char charAt (int index)</code>	הפעולה מקבלת אינדקס כפרמטר. מחזירה את התו הנמצא במקום הזה במחרוזת. מקומו של התו הראשון במחרוזת הוא 0
<code>int compareTo (String anotherString)</code>	הפעולה עורכת השוואה לקסיקוגרפית בין המחרוזת שהתקבלה כפרמטר והמחרוזת הנוכחית...
<code>int compareToIgnoreCase (String str)</code>	הפעולה עורכת השוואה לקסיקוגרפית בין המחרוזת שהתקבלה כפרמטר והמחרוזת הנוכחית... תוך התעלמות מהשוני בין אותיות גדולות וקטנות
	...

ט.2. מערכים

בעבר למדתם על מערכים של טיפוסים בסיסיים. למשל על מנת ליצור מערך של מספרים שלמים בגודל 5, כתבתם:

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

בצורה זו נוצר מערך שכל תא בו מאוחסן המחדל של ג'אווה. במקרה של מספרים שלמים, ברירת המחדל היא 0.

ניתן ליצור מערכים גם ללא שימוש ישיר בפעולה `new`, אם הערכים של איברי המערך ידועים מראש:

```
int[] arr = {1,2,3};
```

מערך הוא עצם שניתן לאחסן בו ערכים בסיסיים או עצמים. על מנת ליצור מערך של קופסאות בגודל 5, נכתוב:

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

מערך זה מכיל חמישה תאים שבכל אחד מהם ניתן לאחסן קופסה. ברגע הגדרת המערך הקופסאות עדיין אינן קיימות.

על מנת לאתחל את המערך יש צורך ליצור עצמים, ואת ההפניות אליהם להציב בתוך תאי המערך:

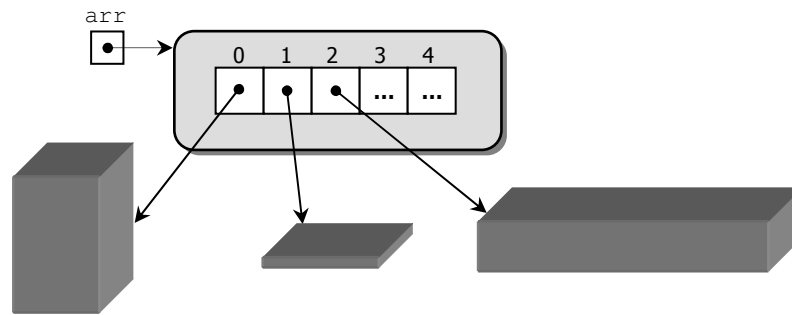
```
arr[0] = new Box(1, 2, 5);
```

```
arr[1] = new Box(3, 2, 1);
```

```
arr[2] = new Box(2, 2.5, 2);
```

...

עתה מוצבים בשלושת התאים הראשונים במערך עצמים מטיפוס Box :



נכתוב תוכנית המחשבת את הנפח הכללי של כל הקופסאות השמורות במערך :

```
double sumVol = 0;
for(int i=0; i<arr.length; i++)
    sumVol = sumVol + arr[i].getVolume();
System.out.println(sumVol);
```

הגישה לעצמים השמורים במערך נעשית בעזרת הכתיבה המקובלת `arr[i]`. אך הפעם בתאים אלה שמורים עצמים, ואנו יכולים לבקש מהם להפעיל כל אחת מהפעולות שלהם, למשל:

```
arr[i].getVolume()
```

נחזור ונעמיק בנושא מערכים של עצמים בפרק 6.

? כתבו תוכנית המגדירה מערך של עשר מחרוזות הנקלטות מהמשתמש, ומדפיסה אותן.

התוכנית תמייין את המערך בסדר אלפביתי עולה ותדפיס אותו לאחר המיון. היעזרו בפעולה

`compareTo(...)` המוגדרת בממשק המחלקה `.String`.

י. סיכום

- תהליך התכנות בשפות מונחות עצמים כולל כתיבת מחלקות, יצירת עצמים ממחלקות אלה והפעלתם באופן שיביא למילוי מטלה שהוגדרה מראש.
- לעצם יש מצב וכן פעולות שהוא יכול לבצע.
- כדי ליצור עצם יש לזמן פעולה בונה שלו בעזרת המילה השמורה `new`. יוצאי דופן הם שניים: עצמים מטיפוס המחלקה `String` שעבורה ניתן לוותר על השימוש ב-`new`, ומערכים שניתנים ליצירה בעזרת רשימת אתחול.
- מחלקה היא גם טיפוס. ניתן להכריז על משתנים מטיפוס זה. כל עצם הנוצר מהמחלקה הוא מטיפוס זה, וניתן לשומו ולפנות אליו בעזרת הפניה הנשמרת במשתנה מטיפוס המחלקה.
- אחרי יצירת עצם ניתן להפעיל את פעולותיו באמצעות שיטת סימון-הנקודה.
- כדי לדעת מהן הפעולות המוגדרות עבור עצם מסוים, יש לפנות לממשק של המחלקה. הממשק נוצר מתוך תיעוד המחלקה ומופיעים בו תיאור תמציתי של העצמים של המחלקה, כותרות הפעולות וכן תיאור קצר של כל פעולה. הממשק אינו מספק מידע על ייצוג מצבם של העצמים ועל אופן המימוש של הפעולות. בכך מיישם הממשק את עקרון ההפרדה בין ממשק למימוש.
- בשפת ג'אווה קיים אוסף של מחלקות מוכנות המאוגדות בחבילות. כדי להשתמש במחלקה יש צורך "לייבא" את החבילה שהיא נמצאת בה, וזה נעשה באמצעות המילה השמורה `import`. אוסף הממשקים של מחלקות אלה נקרא `JavaAPI`.

מושגים

import	"לייבא" (חבילה)
reference	הפניה
method header	כותרת פעולה
instance	מופע
class	מחלקה
interface	ממשק (API)
dot notation	סימון-נקודה
object	עצם
method	פעולה
constructor	פעולה בונה

פרק 2 דף עבודה מס' 1 קופסה צבעונית

לפניכם ממשק הדומה מאוד לממשק Box שהוצג בפרק בהבדל אחד: מאפייני המצב של קופסה זו כוללים גם את צבעה.

א. השלימו את הכותרות המתאימות לתיאור הפעולות. הפעולות המפורטות הן פעולות נוספות לפעולות המופיעות בפרק, פרט לפעולה הבונה ולפעולה המחזירה מחרוזת תיאור. שתי פעולות אלה השתנו בעקבות הרחבת האפיון של מצב הקופסה:

ColoredBox (double length, double width, double height, String color)	פעולה בונה של המחלקה קופסה. הפעולה מקבלת 4 פרמטרים: אורך, רוחב, גובה וצבע, ומאתחלת את מצב הקופסה החדשה לפי ערכי הפרמטרים
השלימו כאן	הפעולה משנה את צבע הקופסה הנוכחית לפי הפרמטר
השלימו כאן	הפעולה מחזירה את צבע הקופסה הנוכחית
השלימו כאן	הפעולה מחזירה מחרוזת המתארת את הקופסה

ב. מדוע הפעולה הבונה השתנתה?

ג. כתבו פעולה ראשית המבצעת את המשימות הבאות:

- בונה שתי קופסאות שונות
- משנה את צבעה של הקופסה הראשונה
- משנה את צבעה ואת רוחבה של הקופסה השנייה
- מדפיסה מחרוזת המתארת את הקופסאות לפני השינוי ואחריו.

בהצלחה!

פרק 2 דף עבודה מס' 2

דלי – Bucket

ממשק המחלקה Bucket

המחלקה מגדירה דלי שהוא בעל קיבולת. הדלי יכול להכיל כמות מים כלשהי עד קיבולת זו.

Bucket (int capacity)	הפעולה בונה דלי ריק שקיבולתו היא הפרמטר .capacity הנחה : קיבולת הדלי היא מספר אי-שלילי
void empty()	הפעולה מרוקנת את הדלי הנוכחי
boolean isEmpty()	הפעולה בודקת את מצב הדלי. אם הדלי הנוכחי ריק היא מחזירה "אמת", ואם לא היא מחזירה "שקר"
void fill (double amountToFill)	הפעולה מקבלת כפרמטר כמות של מים וממלאת את הדלי הנוכחי בכמות זו. אם כמות המים היא מעבר לקיבולת הדלי, הדלי מתמלא ויתר המים נשפכים החוצה. הנחה : כמות המים היא מספר אי-שלילי
int getCapacity()	הפעולה מחזירה את הקיבולת של הדלי הנוכחי
double getCurrentAmount()	הפעולה מחזירה את כמות המים הקיימת בדלי הנוכחי
void pourInto (Bucket bucketInto)	הפעולה מעבירה את כמות המים המקסימלית האפשרית מהדלי הנוכחי לדלי שהתקבל כפרמטר
String toString()	הפעולה מחזירה מחרוזת המתארת את הדלי הנוכחי בצורה הבאה : The capacity: <capacity> The current amount of water: <current amount of water>

מה עליכם לעשות?

1. בתרגיל זה עליכם לעקוב באופן תיאורטי אחרי הקוד בפעולה הראשית ולציין מה יודפס בכל שלב:

```
public static void main(String[] args)
{
    Bucket b1 = new Bucket(5);
    Bucket b2 = new Bucket(8);
    b1.fill(7);
    b2.fill(1);
    b2.empty();
    System.out.println (b1);
    System.out.println (b2);
    b1.pourInto (b2);
    System.out.println (b1);
    System.out.println (b2);
}
```

בהצלחה!

פרק 2 דף עבודה מס' 3 דלי גרפי

ממשק המחלקה הגרפית Bucket

לרשותכם מחלקה גרפית בשם Bucket. כל דלי מטיפוס המחלקה מתאפיין בקיבולת ובשם. מכיוון שמדובר במחלקה גרפית, כל שימוש והפעלה של הדלי מוצגים באופן ויזואלי על המסך, ולכן לא קיימת פעולת toString() בממשק המחלקה. המחלקה מופיעה בספריית unit4 העומדת לרשותכם. לצורך שימוש במחלקה דלי עליכם לכתוב את השורה הבאה בראש התוכנית:

```
import unit4.bucketLib.Bucket;
```

לפניכם ממשק המחלקה הגרפית:

Bucket (int capacity, String name)	הפעולה בונה דלי ריק שקיבולתו היא הפרמטר capacity ושמו הוא הפרמטר name
void empty()	הפעולה מרוקנת את הדלי הנוכחי
boolean isEmpty()	הפעולה בודקת את מצב הדלי. אם הדלי הנוכחי ריק היא מחזירה "אמת", ואם לא היא מחזירה "שקר"
void fill (double amountToFill)	הפעולה מקבלת כפרמטר כמות של מים וממלאת את הדלי הנוכחי בכמות זו. אם כמות המים היא מעבר לקיבולת הדלי, הדלי מתמלא ויתר המים נשפכים החוצה. הנחה : כמות המים היא מספר אי-שלילי
int getCapacity()	הפעולה מחזירה את הקיבולת של הדלי הנוכחי
double getCurrentAmount()	הפעולה מחזירה את כמות המים הקיימת בדלי הנוכחי
void pourInto (Bucket bucketInto)	הפעולה מעבירה את כמות המים המקסימלית האפשרית מהדלי הנוכחי לדלי שהתקבל כפרמטר

מה עליכם לעשות?

1. כתבו תוכנית ראשית המגדירה מערך של 6 דליים ריקים, בקיבולות זהות. מלאו את הדלי הראשון כולו במים. מלאו את הדלי השני במחצית תכולתו של הראשון. מלאו את השלישי במחצית תכולת הדלי השני וכך הלאה. הריצו את התוכנית שכתבתם (שימו לב שכאשר אתם משתמשים במחלקה הגרפית, יש לכל דלי שם ועליכם לספק את השם בעת הפעלת הפעולה הבונה. שמו של הדלי בתוכנית שהתבקשתם לכתוב יכול להיות מספרו הסידורי).
2. כתבו תוכנית הקולטת מספר טבעי n. התוכנית מייצרת n דליים שקיבולת כל אחד מהם היא 20 ליטר. כל דלי ימולא בכמות מים אקראית במספר ליטרים ממשי שערכו בין 0-20. לאחר מכן, התוכנית תמלא את הדליים כך שבכולם תהיה לבסוף הכמות של המים שהייתה מלכתחילה בדלי המלא ביותר.
3. כתבו תוכנית הקולטת מספר טבעי n ויוצרת n דליים שקיבולתם אקראית: בין 0-10 ליטר לדלי. שמות הדליים יכולים להיות מספריהם הסידוריים. הדליים ימולאו בכמויות מים אקראיות, שלא יהיו גדולות מקיבולת הדלי. על התוכנית להדפיס את כמויות המים שבדליים, ממוינים לפי סדר עולה.

בהצלחה!

פרק 2 דף עבודה מס' 4

מתוך JavaAPI : המחלקה Point

בתרגיל זה נשתמש באחת המחלקות המוכנות בשפת ג'אווה ששמה Point. אנו נביא כאן ממשק חלקי שלה. ממשק מלא קיים ונגיש דרך סביבת העבודה.

המחלקה Point (מתוך JavaAPI)

המחלקה Point מגדירה נקודה דו-ממדית.

Point (int x, int y)	פעולה בונה של המחלקה. מאתחלת נקודה חדשה בהתאם לפרמטרים x ו-y
void move (int x, int y)	הפעולה מקבלת כפרמטרים ערכי x ו-y חדשים ומזיזה את הנקודה בהתאם
void translate (int dx, int dy)	הפעולה מוסיפה לערך ה-x של הנקודה את הפרמטר dx ולערך ה-y של הנקודה את הפרמטר dy
String toString()	הפעולה מחזירה מחרוזת המתארת את הנקודה

על מנת להשתמש במחלקה זו עלינו "לייבא" אותה. המחלקה Point נמצאת בחבילה הנקראת java.awt. חבילה זו מכילה את כל המחלקות הגרפיות של שפת ג'אווה. כדי "לייבא" אותה עלינו לכתוב בתחילת התוכנית (לפני הגדרת המחלקה):

```
import java.awt.Point;
```

מה עליכם לעשות?

א. עליכם ליצור שתי נקודות שונות ולהדפיס אותן. לאחר מכן עליכם להזיז את הנקודות למקום חדש: פעם אחת באמצעות הפעולה move, ופעם אחת באמצעות הפעולה translate, ולהדפיס אותן אחרי כל הזזה. הגדירו במילים מה השוני בין שתי פעולות ההזזה.

ב. 1. מה לדעתכם יקרה כאשר נכתוב את שתי השורות הבאות:

```
Point p;  
p.move (3, 4);
```

2. בדקו בעזרת הפעולה הראשית האם התוצאה היא כפי שציפיתם.

מהצ'חה!

פרק 2 דף עבודה מס' 5 מתוך JavaAPI : המחלקה String

בתרגיל זה תשתמשו במחלקה נוספת הקיימת ב-Java, המחלקה String. כיוון שהמחלקה כלולה בחבילה java.lang אין צורך לייבא אותה באופן מפורש.

המחלקה String

לפניכם ממשק חלקי של המחלקה :

<code>char charAt (int index)</code>	הפעולה מקבלת אינדקס כפרמטר. מחזירה את התו היושב במקום הזה במחרוזת. מקומו של התו הראשון במחרוזת הוא 0
<code>boolean endsWith (String suffix)</code>	הפעולה מקבלת מחרוזת כפרמטר. מחזירה "אמת" אם מחרוזת הפרמטר מופיעה כתת-מחרוזת בסוף המחרוזת המבצעת את הפעולה, אחרת מחזירה "שקר"
<code>String replace (char oldChar, char newChar)</code>	הפעולה מקבלת כפרמטרים שני תווים : <code>oldChar</code> ו- <code>newChar</code> , ומחזירה מחרוזת חדשה שבה כל המופעים של <code>oldChar</code> מוחלפים על ידי המופעים של <code>newChar</code>
<code>String toLowerCase()</code>	הפעולה מחזירה מחרוזת חדשה שבה כל התווים נכתבים באותיות קטנות
<code>String toUpperCase()</code>	הפעולה מחזירה מחרוזת חדשה שבה כל התווים נכתבים באותיות גדולות

שימו לב : המחרוזות שמפעילה את הפעולות **לעולם** אינה משתנה. כאשר נדרש שינוי היא מחזירה מחרוזת חדשה ובה השינוי. לדוגמה :

```
String str = "Moshe";
str.toUpperCase();
System.out.println(str);
```

ההדפסה של הקוד היא :

Moshe

כיוון שאין שינוי במחרוזת המקורית.

לעומת זאת הקוד הבא :

```
String str1= "Moshe";
String str2 = str1.toUpperCase();
System.out.println(str2);
```

MOSHE

ידפיס :

מה עליכם לעשות?

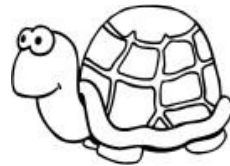
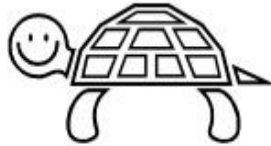
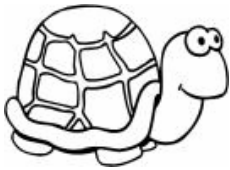
א. מה יודפס במהלך הפעולה הראשית הבאה:

```
public static void main(String[] args)
{
    String s1 = "Moshe";
    String s2 = "Avraham";
    System.out.println(s1.charAt (2));
    System.out.println(s2.toUpperCase());
    System.out.println(s2.replace ('a', 'o'));
    System.out.println(s2);
}
```

ב. פתחו את ה-JavaAPI של המחלקה String בסביבת העבודה. כתבו תוכנית הקולטת 5 שמות. עבור כל שם, אם הוא מסתיים ב"el", התוכנית תדפיס את כל השם באותיות גדולות. אם השם מסתיים ב"iam" התוכנית תדפיס את כל השם באותיות קטנות. בכל מצב אחר, התוכנית כלל לא תדפיס את השם.

הנה! חנה!

פרק 2 דף עבודה מס' 6 לרקוד עם צבים



הקדמה

בתרגיל זה נצייר ציורים שונים על משטח גרפי באמצעות צבים בדומה ל-LOGO. כאשר יוצרים צב, הוא ממוקם במרכז המשטח הגרפי, כשפניו צפונה. כעת, ניתן לתת לצב "פקודות" על מנת שיצייר על המשטח הגרפי. ניתן לשנות את מיקומו וכיוונו של הצב במטרה לגרום לו לנוע על המשטח. לצב יש זנב, אותו ניתן להרים ולהוריד ובאמצעותו ניתן לצייר על המשטח הגרפי. ציור על המשטח הגרפי יתאפשר רק אם זנבו של הצב למטה.



לרשותכם מחלקה קיימת בשם Turtle ממנה ניתן ליצור עצמים מסוג צב, שבאמצעותם ניתן לצייר על המשטח הגרפי. כדי להשתמש במחלקה עליכם לייבא את החבילה `unit4.turtleLib` לכל תוכנית שבה תשתמשו בצבים. חלק מהפעולות שיכול עצם מטיפוס צב לבצע, המוגדרות על ידי המחלקה Turtle, מתוארות בממשק הבא:

Turtle()	הפעולה בונה עצם מסוג צב הממוקם במרכז, פניו צפונה וזנבו למעלה.
void moveBackward (double x)	הפעולה מזיזה את הצב x צעדים אחורה
void moveForward (double x)	הפעולה מזיזה את הצב x צעדים קדימה
void tailDown()	הפעולה מורידה את זנב הצב
void tailUp()	הפעולה מרימה את זנב הצב
void turnLeft (double x)	הפעולה מפנה את פני הצב x מעלות נגד כיוון השעון
void turnRight (double x)	הפעולה מפנה את פני הצב x מעלות עם כיוון השעון

תוכנית לדוגמה

התוכנית הבאה מציירת בעזרת הצב t1, מלבן שרוחבו 50 ואורכו 100 :

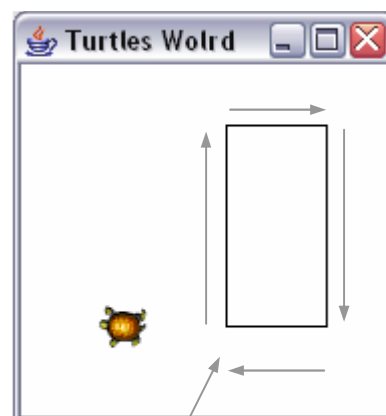
```
import unit4.turtleLib.Turtle;

public class TurtleDrawRectangle
{
    public static void main(String[] args)
    {
        Turtle t1 = new Turtle();

        t1.tailDown();

        t1.moveForward(100);
        t1.turnRight(90);
        t1.moveForward(50);
        t1.turnRight(90);
        t1.moveForward(100);
        t1.turnRight(90);
        t1.moveForward(50);

        t1.tailUp();
        t1.moveForward(50);
    }
}
```

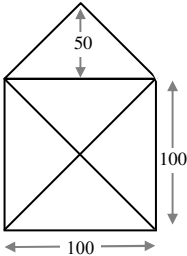
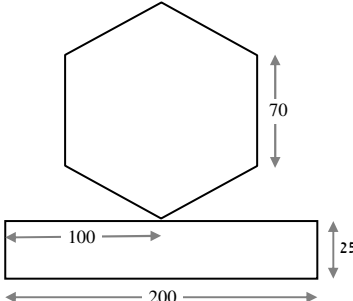


נקודת
ההתחלה

מה עליכם לעשות?

א. ענו על השאלה: מה מגדיר את מצבו של הצב?

ב. עליכם לכתוב שתי תוכניות:

2. תוכנית בשם House.java שתצייר בית	1. תוכנית בשם Hexagon.java שתצייר משושה על תיבה
	

הוסיפו בראש התוכנית שלכם את הפקודה `import unit4.turtleLib.Turtle`, וכך תוכלו לעבוד עם הצבים. כתבו את שתי התוכניות הנדרשות: `House.java` ו-`Hexagon.java`:

i. התוכנית `Hexagon.java` תבנה משושה **משוכלל** על גבי תיבה לפי המידות המפורטות בציוור (במשושה משוכלל כל הזוויות וכל הצלעות שוות זו לזו).
שימו לב כי בעת ציור המשושה, הצב מבצע סיבוב שלם ולכן במהלך ציור משושה יסתובב 360° .

ii. שימו לב שאם תחזרו על שרטוט קו יותר מפעם אחת – הוא יבלוט בעוביו ולכן עליכם להימנע מכך. כל הקווים צריכים להיות שווים בעוביים.

בהצלחה!

פרק 2 דף עבודה מסי 7 שוליית הקוסם

רקע

הערה: אם אינכם מכירים את סיפורו של שוליית הקוסם, חפשו באינטרנט את המחרוזת "שוליית הקוסם".

לרוע המזל, הצליח שוליית הקוסם להציף את חדר העבודה של הקוסם במים. למרבה המזל, עומדים לרשותו של השוליה שני דלי קסם.

הדלי הראשון, יכול למלא עצמו מהמים שבחדר, אך אינו יכול לצאת מהחדר.

הדלי השני, אינו יכול לגרוף ישירות מים מהחדר, אולם הוא יכול להתמלא מהדלי הראשון, ולצאת אל החצר לרוקן את עצמו.

בתרגיל זה עליכם לעזור לשוליית הקוסם לבצע את המשימה.

כדי לרוקן את החדר מהמים שהציפו אותו ניצור הדמיה של הסיפור. לשם כך נבנה את החדר ושני דליים בגדלים מתאימים. שלב זה נקרא "מידול הסיפור", שכן אנו מגדירים מודולים שייצגו את הישויות האמיתיות בתוכנית שלנו. בשלב הבא תציע התוכנית פתרון לבעייתו של שוליית הקוסם: היא תרוקן את החדר על ידי שימוש נכון בשני הדליים. כדי לבדוק ולהתרשם מתוכניתכם באופן ויזואלי ומעניין, השתמשו במחלקה הגרפית Bucket, שאותה הכרתם בדף עבודה מספר 3.

הבה נתחיל בעבודה!

בניית הסיפור

לצורך הדמיית הסיפור נשתמש בשלושה דליים. שניים מהם מתוארים ישירות בסיפור, אך שימו לב שגם החדר יכול להיות מיוצג על ידי דלי! תנו דעתכם: החדר גם הוא מכל בגודל מסוים היכול להכיל מים, שיכולים למלא את החדר ושניתן לרוקנם ממנו. לכן אין צורך בהגדרת מחלקה חדשה ואפשר להסתכל על החדר כעל דלי. נקרא ל"דלי" זה בשם room, לעומת הדליים האחרים שייקראו בשמות של דליים – bucket1 ו-bucket2.

אפיון אחד במצבו של דלי יהיה ה"שם" (name). כיוון שהמחלקה דלי הנוכחית היא גרפית, יוצג ה-name של כל דלי לידו על המסך.

א. הנתונים

כדי לפתור את הבעיה אנו זקוקים לכמה נתונים:

1. קיבולת החדר (כמות המים שהחדר יכול להכיל).
2. קיבולת הדלי הראשון.
3. קיבולת הדלי השני.
4. כמות המים ההתחלתית בחדר.

את קיבולת החדר ניתן לקבוע מראש בתוכנית (או לקלוט מהמשתמש, כרצונכם). את שלושת הנתונים הנוספים עלינו לקלוט מהמשתמש. עליכם לבדוק שהקלט שתקבלו עבור כל אחד מהנתונים, אכן תקין. דרישות התקינות מפורטות בהמשך דף עבודה זה. לשם פשטות העבודה, נתחיל לכתוב את התוכנית ללא בדיקות התקינות, ונתפנה להוסיפן רק בשלב שני (אם יישאר לכם זמן לעבודה במעבדה).

ב. בניית העצמים

ניצור שלושה עצמים מטיפוס Bucket. הדלי הראשון ייצג את החדר ושני הדליים הנוספים ייצגו את דליי הקסם של שוליית הקוסם. הקפידו לבנות את הדליים עם קיבולות כמתואר בסעיף הקודם.

כפי שתוכלו לראות, בממשק של המחלקה Bucket, המופיע בהמשך, הפעולה הבונה של מחלקה זו בונה דליים ריקים. את שני דליי העזר השאירו לעת עתה ריקים. את החדר, לעומת זאת, מלאו בכמות המים ההתחלתית שבחר המשתמש, תוך שימוש בפעולות של המחלקה "דלי".

פתרון הבעיה

כעת תוכלו להפעיל את הדליים שיצרתם כדי לרוקן את המים מהחדר. כזכור, המשימה היא לרוקן את כל המים מהחדר. מהחדר מותר להעביר מים רק לדלי הקסם הראשון. ואולם, לרוקן מים החוצה מותר רק מדלי הקסם השני. כלומר, הוצאת המים מהחדר תיעשה בשלבים: מים יועברו מהחדר לדלי הראשון, לאחר מכן הם יועברו מהדלי הראשון לדלי השני, ולבסוף ירוקנו מהדלי השני לחצר. על פעולת העברת מים מהדלי הראשון לדלי השני נחזור ככל הדרוש, עד שהדלי הראשון יתרוקן כליל, ואז נמלא אותו שוב במים מהחדר ונרוקן את הדלי השני לחצר, וכך הלאה, עד לריקונו המוחלט של החדר. העברת המים מהחדר לדלי הראשון אפשרית אך ורק אם הדלי הראשון ריק לחלוטין.

לשם פתרון הבעיה השתמשו בפעולות השונות של המחלקה דלי, כפי שהן מופיעות בממשק המחלקה Bucket, שהיא חלק מהחבילה unit4.bucketLib. כמובן שעליכם לציין בראש התוכנית:

```
import unit4.bucketLib.Bucket;
```

הרצת התוכנית

עם ריצת התוכנית ייפתח ממשק גרפי שיאפשר לכם לעקוב אחרי הפתרון שכתבתם. ודאו שאכן הצלחתם לרוקן את כל כמות המים שבחדר, מבלי לחרוג מהדרישות שהוגדרו.

עכשיו, אם עוד נותר לכם זמן מעבדה, השלימו את בדיקות התקינות שדילגנו עליהן קודם:

דרישות התקינות של הקלט

במהלך דף עבודה זה נדרשתם לבקש מהמשתמש שלושה או ארבעה נתונים: קיבולת שני הדליים, כמות המים ההתחלתית בחדר ואולי אף את קיבולת החדר. עבור כל אחד מהנתונים, עליכם לבדוק האם הקלט שקיבלתם תקין. אם הקלט אינו תקין, עליכם להדפיס הודעת שגיאה מתאימה, ולחזור ולבקש קלט חדש עד אשר יתקבלו נתונים תקינים. דרישות התקינות הן:

- כמות המים ההתחלתית בחדר צריכה להיות קטנה או שווה לקיבולת החדר. כמו כן, על כמות זו להיות גדולה או שווה ל-0.
- קיבולתו של הדלי הראשון צריכה להיות גדולה ממש מ-0.
- קיבולתו של הדלי השני צריכה להיות גדולה ממש מ-0.

בהצלחה!