

פרק 3

מחלקות ועצמים

יותם נכנס הביתה, זרק את התרמיל על המיטה ופתח את המקרר בעצבנות. "שוב אין מה לאכול!", רטן לעצמו, ורק אז הבחין במכתב שחיכה לו על השולחן הקטן ליד המקרר. הוא פתח את המכתב בסקרנות, וסקרנותו גברה ככל שהמשיך לקרוא. המכתב היה מג'ון, חברו האנגלי, הרפתקן חסר תקנה. ג'ון, כך מסתבר, קיבל עבודה חדשה ממשרד התיירות האקוודורי באיי גלפאגוס.

"כידוע, באי יש בעלי חיים רבים ומגוונים. מדענים החוקרים את האי", כך הסביר ג'ון, "מנסים לנבא את התפתחות אוכלוסיית בעלי החיים באי. על סמך העובדות כיום, מנסים המדענים לשער אילו בעלי חיים ישרדו באי בעתיד, וכיצד ישפיעו תנאים שונים (כגון התחממות כדור הארץ או הרחבת התיירות) על אוכלוסיית האי. לשם כך הם החליטו להזמין תוכנת מחשב, שתבצע סימולציה (הדמיה) של פעילויות שונות באי, ותסייע להם בפתרון הבעיה".

ג'ון התבקש לנסוע מיידית לאי כדי לאסוף נתונים, ולכתוב את תוכנת המחשב. "השלב הראשון של הפרויקט", הסביר ג'ון, "הוא ליצור מאגר גדול של מחלקות, המייצגות את בעלי החיים השונים באי. לאחר מכן, נוכל להשתמש במחלקות אלו בכדי לכתוב תוכניות שידמו את התפתחות האוכלוסיות, ויחשבו את השפעת התנאים השונים עליהן. אולם", המשיך ג'ון, "זוהי משימה גדולה ומורכבת".

לכן הוא החליט להיעזר בכמה מתכנתים. בעוד הוא ירכז את הפרויקט, תוכל תוכניתו להשתמש במחלקות שאותן יכתבו מתכנתים שונים בצוות.

נשימתו של יותם כמעט נעתקה כשקרא את השורה הבאה. "רצייתי לברר", כתב ג'ון, "אם תוכל לכתוב את אחת המחלקות. אני יודע שאתה לומד מחשבים, וחשבתי שתהנה מהניסיון. מדובר במחלקה המייצגת את האיגואנה. זוהי לטאה ענקית ויפהפייה החיה באי. אני אסביר לך אילו שיטות ברצוני שהאיגואנה תבצע, ואתה תתכנת את המחלקה לפי הבנתך, ובתנאי שתוכל לבצע את השיטות הנדרשות לי. מה דעתך?"

השאלה הייתה מיותרת. יותם הוציא מהתרמיל עט ונייר, והחל בתכנון המחלקה החדשה.

א. מחלקה (Class)

לקרוא לילד בשמו

הבה נבחן את המשימה שניצבה בפני יותם. עליו לתכנן מחלקה שהיא התבנית שלפיה יבנה ג'ון את כל העצמים מסוג איגואנה בתוכניתו.

תחילה, היה עליו לבחור שם למחלקה. בתחילה חשב לקרוא למחלקה YotamsCoolClass, וכך לזכות בתהילת עולם. אולם אז נזכר כי אין זה מקובל. בג'אווה כמו בכל שפה, ישנן **מוסכמות**

(conventions) לגבי סגנון הכתיבה בשפה. מוסכמות אלו, הן כללים שעל כל מתכנת בשפה לציית להם.

המוסכמות המקובלות בקשר לשמות של מחלקות, הן:

- לקרוא למחלקה בשמו של הסוג או הטיפוס שאותו היא מגדירה, מאחר והמחלקה היא התבנית הכללית של אובייקטים מסוג מסוים. לדוגמה: `class Iguana`.
 - מקובל להתחיל את שם המחלקה באות גדולה, לדוגמה: `class Turtle` (הנפתח באות הגדולה T).
 - ג'אווה מעודדת שימוש בשמות בעלי משמעות. כדי לאפשר קריאה נוחה של קוד (אפילו אם הוא מכיל שמות מורכבים וארוכים), ייכתב שם המחלקה כך שכל מילה חדשה תתחיל באות גדולה.
- לדוגמה: שמה של המחלקה "תולעת ורודה שלה נקודות כחולות", ייכתב כך:

```
class PinkWormWithBlueDots
```

במהלך פרק זה עוד נתקל במוסכמות רבות.

מדוע עלינו להיות כפופים למוסכמות אלו? התוכנית יכולה לרוץ גם אם נחרוג מהמוסכמות. אך מוסכמות חשיבות מרובה, והשימוש בהן הוא הכרחי כדי ליצור קוד מובן, קריא ונוח לשימוש. הפרויקט שבו משתתף יותם הוא דוגמה טובה לצורך במוסכמות. התוכנית הסופית תכיל מספר רב של מחלקות. העובדה שכל מחלקה תיכתב בהתאם לאותן מוסכמות תקל על השימוש במחלקות השונות והרבות. כמו כן, עדכון התוכנית בעתיד על ידי מתכנתים אחרים לא יהיה מסובך.

ההכרזה על המחלקה

כדי לבנות מחלקה חדשה יש להכריז עליה באופן הבא:

```
public class NameOfClass{  
    ...  
}
```

השורה המופיעה בראש ההכרזה, נקראת **כותרת המחלקה (class heading)**.

המילה השמורה `public` פותחת את כותרת המחלקה. מילה זו מציינת כי השימוש במחלקה יהיה פומבי, כלומר חופשי לשימוש כל מחלקה אחרת שמעוניינת בכך. כדי לאפשר כימוס של עצמים והסתרת מידע מחד גיסא, אך חשיפה של המידע במקומות שבו הוא חיוני מאידך גיסא, מאפשרת ג'אווה למתכנת לבחור בהרשאות גישה שונות. הסבר מפורט יותר על נושא הרשאות הגישה השונות מופיע בפרק 5. המילה השמורה `class` מציינת כי זוהי הכרזה על מחלקה. אחריה יופיע שם המחלקה וסוגריים מסולסלים, שבתוכם ייכתב גוף המחלקה:

```
public class Iguana{  
    ...  
    // כאן ייכתב גוף המחלקה  
    ...  
}
```

אחד העקרונות החשובים בתכנות הוא תיעוד כל התוכנית והסבר נאות שלה. כדי לתעד את התוכנית כהלכה, מקובל להוסיף בראש המחלקה הערה כללית המתארת אותה:

```
/**
```

```
* This class simulates an Iguana.  
* It is part of the Galapagos Islands project, directed by John Shore.  
* author: Yotam Cohen.  
*/
```

```
public class Iguana{
```

```
    ...
```

```
}
```

קובץ חדש למחלקה החדשה

את המחלקה החדשה שמר יותר בקובץ הנקרא: Iguana.java. כל מחלקה בג'אווה, 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 של השפה. ההתאמה בין שמות הקבצים לשמות המחלקות היא שמאפשרת למהדר לקשר ביניהם, ולמצוא באיזה קובץ שמורה כל מחלקה.

ב. תכונות

בחירת תכונות

אנו יודעים כי כל מחלקה מורכבת מתכונות (attributes) ומשיטות (methods) המאפיינות את המחלקה.

התכונות הן רשימת המאפיינים של מופעי המחלקה. כדי להגדיר את התכונות, יש לבחור תחילה את התכונות שבהן אנו מעוניינים. על התכונות שנבחר לאפיין את העצם, מתוך התייחסות לבעיה התכנותית שאותה אנו מנסים לפתור. במקרה של המחלקה איגואנה, התכונות הללו יכולות להיות צבעה של האיגואנה, משקלה, המזון המועדף עליה וכן הלאה. התכונות יהיו הדבר הראשון שאותו נגדיר בכל מחלקה.

הצהרה על תכונות

התכונות מיוצגות על ידי משתנים, לכן נצהיר עליהן בדיוק כפי שלמדנו להצהיר על משתנים בפרק הקודם.

```
/**
 * This class simulates an Iguana.
 * It is part of the Galapagos Islands project, directed by John Shore.
 * author: Yotam Cohen
 */
public class Iguana{
    //the attributes of the class:
    private int      numOfChildren;
    private double   weight;
    private boolean  isFriendly;
    ...
}
```

ההצהרה על תכונה נפתחת במילה השמורה *private* (פרטי). המילה *private* מורה על כך שהתכונה היא מאפיין פרטי של המחלקה, ומונעת ממחלקות אחרות לשנות את ערכה או לקרוא אותו ישירות. גם כאן, הוספת מילה זו נחוצה שכן זו אינה ברירת המחדל של ג'אווה. המילה *private* בדומה למילה *public*, שייכת לנושא הרשאות הגישה בג'אווה (שבו נדון בהרחבה בפרק 5). כמו בכל הצהרה על משתנה, יש לציין את טיפוס התכונה ואת שמה. הטיפוס של התכונה המייצגת את מספר הצאצאים של האיגואנה הוא *int* ושמה הוא `numOfChildren`, ושל התכונה המייצגת את משקלה (שאינו מיוצג במספר שלם בהכרח) הוא *double* ושמה הוא `weight`.

מסתבר שאחת ההבחנות החשובות של המדענים היא בין איגואנות הנוטות להיות חברותיות לכאלו החיות בבידוד. לפיכך הצהיר יותם על תכונת "חברותיות" המיוצגת כמשתנה מטיפוס *boolean*. אם האיגואנה חברותית משתנה זה יקבל את הערך *true*, אחרת יקבל *false*.

המוסכמה שננהיג לגבי שמות התכונות היא המוסכמה עבור שמות משתנים בכלל: השם ייכתב באותיות קטנות, למעט תחילתה של כל מילה פנימית חדשה, שתכתב באות גדולה.

ערכי ברירת המחדל של תכונות

באופן כללי, כאשר מצהירים על משתנים בג'אווה, שום ערך אינו מוצב בהם. אם משתנים אלה הם תכונות, אזי תוך כדי בניית העצם הם מאותחלים לערכי ברירת מחדל, גם אם המתכנת לא מאתחל אותם במפורש (כפי שנראה בהמשך). ערך ברירת המחדל עבור תכונות מטיפוסים בסיסיים המייצגים מספרים (כגון *int* או *double*) הוא אפס, ואילו ערך ברירת המחדל עבור תכונות מהטיפוס *boolean* הוא *false*. למשל, לא הצבנו בתכונה `isFriendly` שום ערך, ובכל זאת ג'אווה הציבה בה באופן אוטומטי את הערך *false*.

למרות האמור לעיל, מומלץ מאוד לא להסתמך על ערכי ברירת המחדל של ג'אווה, ולהציב בתכונות ערכים על פי צורכי התוכנית שלכם. בהמשך הפרק נלמד כיצד לעשות זאת.

פנייה אל תכונות

נסטה לרגע מן הדיון בהגדרת המחלקה, ונתמקד בשימוש במחלקה. לאחר שנגדיר את המחלקה איגואנה, יוכלו מחלקות אחרות להשתמש בה, וליצור בעזרתה מופעים של המחלקה. כאמור בפרק הראשון, לכל מופע שניצור יהיו אותן השיטות המוגדרות במחלקה, אך ערכים נפרדים לתכונות של כל איגואנה. כיצד ניתן לפנות אל איברי האיגואנה? בפרק הקודם ראינו כי כדי לזמן שיטות של עצם בג'אווה, אנו משתמשים בשיטת סימון מיוחדת הנקראת **סימון-הנקודה (dot notation)**. כדי לבקש מעצם להפעיל שיטה, נכתוב את שם האובייקט, משמאלו נקודה, ומשמאלה את שם השיטה המבוקשת.

כדי להתייחס אל תכונה, נשתמש גם כן בסימון-הנקודה:

```
name.member;
```

סימון זה מאפשר לנו להתייחס אל המשתנה המגדיר את התכונה של מופע מסוים. למשל, אם יצרנו איגואנה בשם `igi`, נוכל להתייחס לתכונת המשקל שלה באמצעות הצירוף `igi.weight`:

```
System.out.print("weight is: " + igi.weight);
```

```
igi.weight = igi.weight + 30;
```

חשוב לזכור שלא מכל מחלקה נוכל לבצע פנייה ישירה כזו אל התכונות. מאחר והגדרנו את הרשאת הגישה לתכונות במחלקה Iguana כ-`private`, תהיה גישה זו חסומה למחלקות אחרות. כך נוכל להבטיח כי ערכי התכונות של `igi` לא ישונו בטעות או באופן בלתי מבוקר. את ערכי התכונות נוכל בכל זאת לברר, אם נוסיף למחלקה **שיטה** שתאפשר זאת. דוגמה לשיטה כזו היא השיטה הראשונה שנגדיר בסעיף הבא.

ג. ועכשיו שיטות

החלק השני שממנו מורכבת כל מחלקה הוא השיטות שלה. השיטות הן הפעולות המיוחדות, שאותן נוכל לבקש מכל מופע של המחלקה לבצע. בתור התחלה, ביקש גיון מיותם, להוסיף למחלקה את שלוש השיטות הבאות:

1. שיטה המחזירה את מספר הצאצאים של האיגואנה.

2. שיטה המדמה את פעולת האכילה של האיגואנה. האיגואנה אוכלת מספר נתון של אצות עסיסיות, מה שגורם למשקלה לעלות בהתאם.

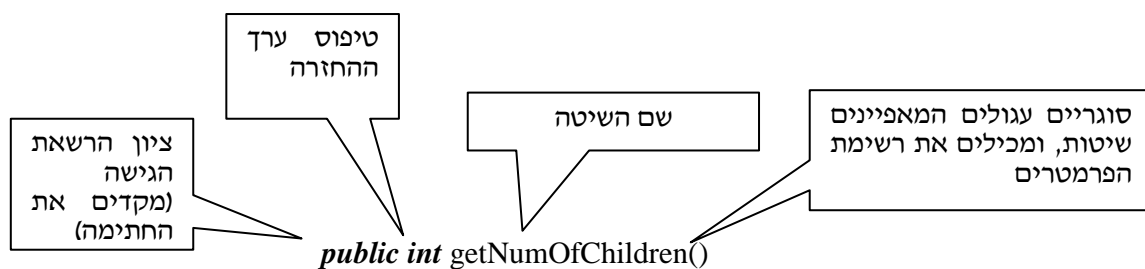
3. שיטה לחישוב מספר הצאצאים החדש. בכל פעם שאיגואנה מסוימת מטילה ביצים, רק חלק מתוכן אכן בוקע. על שיטה זו לקבל את מספר הביצים שהטילה האיגואנה, ואת האחוז מתוכן, שעל פי התנאים הנתונים, עתיד לבקוע. מטרתה של השיטה היא להעריך כמה צאצאים חדשים יבקעו לאיגואנה, ולדמות את הבקיעה בתוכנית.

השיטה מחשבת, אם כן, כמה ביצים בקעו לאיגואנה (על פי הערכת ההדמיה), מעדכנת את מספר הצאצאים של האיגואנה (הווירטואלית), ומחזירה את מספר הצאצאים החדש.

שיטה ראשונה – לתת מבלי לקבל

השיטה הראשונה מחזירה את מספר הצאצאים של האיגואנה. כפי שראינו, על פי רוב עקרון הסתרת המידע מוביל אותנו להגביל את הגישה אל התכונות, על ידי שימוש במילה השמורה *private*. בשל כך נחסמת הגישה הישירה אל התכונות. השיטה שנוסוף תאפשר למופעים של מחלקות אחרות לברר את ערכה של התכונה, אך לא לשנותה.

כל שיטה נפתחת בחתימה (signature):



ממה מורכבת החתימה?

1. טיפוס הערך שאותו מחזירה השיטה – במקרה זה השיטה תחזיר את מספר הצאצאים של האיגואנה, שהוא ערך מטיפוס *int*.
2. שם השיטה – על שם זה לשקף את מה שהשיטה עושה. המוסכמה לגבי שמות השיטות זהה למוסכמה לגבי שמות משתנים. לשיטה זו יותרם בחר את השם `getNumOfChildren`.
3. סוגריים עגולים () ריקים – מסמנים למהדר של ג'אווה כי זוהי שיטה (בניגוד לתכונה). כל שיטה מלווה תמיד בסוגריים בהגדרתה, כמו גם בקריאתה. אם לשיטה יש פרמטרים – הם מופיעים בתוך הסוגריים, אם אין לה פרמטרים (לדוגמה השיטה `getNumOfChildren()` – הסוגריים ריקים).

לפני החתימה, יופיע בכותרת השיטה ציון הרשאת הגישה. מאחר ובמקרה זה השיטה נועדה לשימושן של מחלקות אחרות בתוכנית, בחרנו בהרשאה מסוג *public*. לאחר חתימת השיטה, מוסיפים סוגריים מסולסלים שבתוכם יופיע גוף השיטה. כל שעלינו לעשות בגוף שיטה זו הוא להחזיר את ערכה של התכונה. החזרת ערך כלשהו נעשית בג'אווה על ידי שימוש במילה השמורה *return*. הקוד שכתב יותרם לשיטה נראה כך:

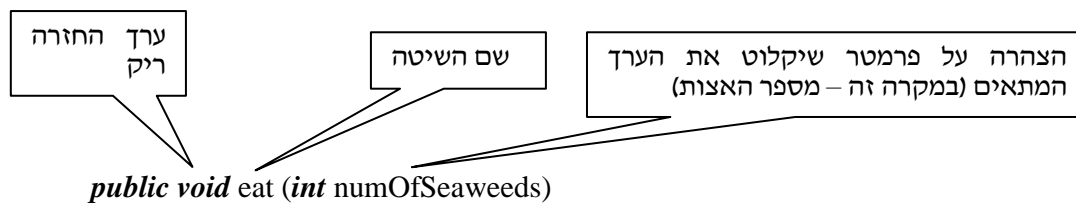
```
/** ... comment... */
public int getNumOfChildren(){
    return this.numOfChildren;
}
```

ודאי הבחנתם שזו הפעם הראשונה שאנו משתמשים במלה *this*. מצורתה המוטה והמודגשת תוכלו להסיק שהיא מלה שמורה בשפת ג'אווה. המלה *this* מייצגת את האובייקט עצמו. כלומר, אם בתוך קטע קוד המממש שיטה, רוצים לפנות לתכונה או לשיטה של האובייקט עצמו משתמשים ב-*this*. כאשר אנו כותבים את הקוד של השיטה, איננו יודעים מה שמו של העצם שיפעיל אותה אך אנו יודעים שזה העצם "הזה", הנוכחי, ולכן קוראים לו *this*.
 עתה, כאשר ירצה ג'ון לברר בתוכניתו את מספר הילדים של האיגואנה איגי, הוא יוכל לברר זאת על ידי הפקודה:

```
int numOfChlidren = igi.getNumOfChildren();
```

השיטה השנייה – לקבל מבלי להחזיר

השיטה השנייה מדמה את פעולת האכילה של האיגואנה. למה הכוונה? בפרויקט שבו משתתף יותר, נכתבת תוכנית סימולציה. סימולציה היא תוכנית המדמה באופן וירטואלי פעילות הנעשית במציאות. כך למשל, אם במציאות איגואנות נוהגות לאכול שלוש אצות בכל בוקר, נרצה שבתוכנית הסימולציה תבצע האיגואנה הווירטואלית איגי פעולה מקבילה. לשם כך, עלינו לכתוב שיטה שתבקש מהאיגואנה הווירטואלית איגי "לאכול". חתימת השיטה נראית כך:



כמו בשיטה הקודמת, את החתימה מקדימה הרשאת הגישה *public*. החלק הבא בחתימת שיטה הוא ערך ההחזרה שלה.

בשפות תכנות מסוימות נהוג לחלק את השיטות השונות לפונקציות ולפרוצדורות. בעוד שפונקציות תמיד מחזירות ערך, הפרוצדורות מבצעות תהליך מסוים, מבלי להחזיר ערך. בג'אווה אין חלוקה מפורשת שכזאת, אולם עדיין ניתן להבחין בין שיטות מהסוג הראשון לשיטות מהסוג השני. השיטה הראשונה שכתב יותר מקבילה לפונקציה (היא מחזירה את מספר הילדים). לעומתה, השיטה השנייה גורמת לאיגואנה לאכול. מטרתה היא לשנות את מצבה של האיגואנה מבלי להחזיר כל ערך. לכן, שיטה זו מקבילה לפרוצדורה.

בג'אווה מסמנים את העובדה שהשיטה אינה מחזירה ערך על ידי שימוש במילה השמורה *void*. מילה זו, שפירושה באנגלית "חֵלֶל" או "ריק", מציינת כי ערך ההחזרה של השיטה הוא ריק, כלומר, אין ערך כזה!

החלק הבא בחתימת השיטה הוא שמה. הפעם בחר יותרם בשם הפשוט: *eat*. אחרונים נותרו הפרמטרים של השיטה. במקרה זה, לשיטה מועבר ערך אחד, שהוא מספר האצות שאוכלת האיגואנה. כאשר נרצה, למשל, לבקש מהאיגואנה שמיל לאכול חמש אצות, נצטרך להעביר לשיטה את הערך 5. אולם כיצד תקלוט השיטה את הערך המועבר? לשם כך, עלינו

להצהיר על משתנה מיוחד, שיקלוט את הערך. משתנה כזה נקרא "פרמטר", ונצהיר עליו תמיד בתוך הסוגריים העגולים שבחתימת השיטה. השיטה השנייה שאותה כתב יותם מופיע פרמטר אחד, והוא משתנה שנועד לציין את מספר האצות שיועבר לשיטה. מאחר ומספר זה הוא תמיד שלם, יותם הגדירו כמשתנה מטיפוס *int*.

מכאן, אנו יכולים להבין את ההצהרה שבתוך הסוגריים:

```
(int numOfSeaweeds)
```

עתה, כאשר נקרא לשיטה על ידי:

```
shmil.eat (5);
```

יועבר הערך 5 לתוך הפרמטר *numOfSeaweeds*, ונוכל לפנות אל הערך דרך פרמטר זה. בעזרת הפרמטר נוכל לעדכן את משקלה של האיגואנה כנדרש, כפי שאכן עשה יותם בשיטתו:

```
public void eat (int numOfSeaweeds){  
    //multiply the number of seaweeds by weight of each seaweed  
    //and add to current weight.  
    this.weight += numOfSeaweeds *WEIGHT_OF_SEAWEED;  
}
```

שימו לב לשימוש בקבוע *WEIGHT_OF_SEAWEED*. קבוע זה מציין את משקלה של כל אצה, ועלינו להגדיר אותו בראש המחלקה, בצמוד לרשימת התכונות:

```
public class Iguana{  
    public final int WEIGHT_OF_SEAWEED = 10;  
    ...  
}
```

את הקבוע המייצג את משקל האצה הגדרנו בעזרת הרשאת הגישה *public*. מאחר ותכונה זו היא קבועה, אין לחשוש שמחלקות אחרות ישנו אותה ללא ידיעתנו.

השיטה השלישית – לתת ולקבל

שיטה זו מקבלת שני פרמטרים: מספר הביצים שהטילה האיגואנה והאחוז מתוכן שצפוי לבקוע. השיטה מדמה את הבקיעה המשוערת, מעדכנת את מספר הצאצאים של האיגואנה בהתאם, ומחזירה את מספר הצאצאים המעודכן של האיגואנה. כעת, כבר יש בידינו הכלים להבין כיצד שיטה זו תהיה בנויה. חתימת השיטה תהיה מורכבת כרגיל משלושה חלקים:

1. טיפוס ערך ההחזרה של השיטה (במקרה זה *int*).
2. שם השיטה (במקרה זה *numOfChildrenAfterHatching*, כלומר מספר הצאצאים לאחר הבקיעה).

3. רשימת הפרמטרים בסוגריים עגולים. במקרה זה הסוגריים יכילו שני פרמטרים: מספר הביצים שהוטלו – פרמטר מטיפוס *int*; והאחוז שבקע מתוכן – פרמטר מטיפוס *double*. כרגיל, נצמיד לחתימה את הרשאת הגישה *public* חתימת השיטה היא אם כן:

```
public int numOfChildrenAfterHatching (int numOfEggsLayed, double percentOfHatching)
```

הפרמטרים מופרדים זה מזה בפסיק. כאשר נרצה לקרוא לשיטה זו, יהיה עלינו להעביר לשיטה שני ערכים המופרדים בפסיק, באופן הבא:

```
igi.numOfChildrenAfterHatching (4 , 25.0);
```

? בהנחה שלאיגי לא היו צאצאים קודמים, מה תחזיר השיטה?

שימו לב כי חשוב מאוד להעביר את הפרמטרים בסדר הנכון. בחתימת השיטה מופיעות הצהרות על שני פרמטרים, שאמורים לקלוט את הערכים המועברים. הערך הראשון שהעברנו (4) במקרה זה, ייקלט במשתנה הראשון, והערך השני (25.0) במשתנה השני. נניח כי ננסה, בטעות, לקרוא לשיטה בהיפוך סדר הפרמטרים:

 `igi.numOfChildrenAfterHatching (25.0 , 4);`

הפעלת השיטה באופן זה תגרום לכך שהערך 25 יוצב בטעות כמספר הביצים שהוטלו, והערך 4, כאחוזים מתוכם שבקעו – מה שכמובן יגרום לשגיאה בתוכנית. במקרה זה, השגיאה היא שגיאת טיפוס (ניסיון להציב את המספר הממשי 25.0 במשתנה מסוג *int*), ולכן תתקבל הודעת שגיאה בעת ההידור.

שגיאה חמורה יותר תיווצר אם שני הפרמטרים הם מאותו הטיפוס (למשל שניהם מטיפוס *int*). במקרה זה, הצבת הערכים בסדר שגוי תוביל לטעות לוגית בתוכנית. את הטעות הזו יהיה קשה הרבה יותר לזהות, מכיוון שלא תתקבל כל הודעת שגיאה מהמהדר.

השיטה המלאה תראה כך :

```
/**
 * A method that receives the number of eggs laid by the Iguana,
 * the percentage of them that hatched, and updates the number of Iguana's
 * children accordingly. It also returns the updated number of children.
 */
public int numOfChildrenAfterHatching (int numOfEggsLayed, double
                                         percentOfHatching){
    //calculate how many eggs actually hatched:
    int eggsHatched = (int)(numOfEggsLayed * percentOfHatching/100);
    //update num of children:
    this.numOfChildren = this.numOfChildren + eggsHatched;
    //return the updated number:
    return this.numOfChildren;
}
```

בשורה הראשונה מחשבת השיטה את מספר הביצים שבקעו בהצלחה, על פי אחוזי הבקיעה הנתונים. אולם, מאחר ומספר זה אינו שלם בהכרח, וברור כי אנו מדברים על מספר שלם של ילדים שנוספו, נשמיט את החלק העשרוני של המספר על ידי ביצוע פעולת ההמרה (casting) שעליה למדנו בפרק הקודם.

אם נזמן את השיטה הבאה (מתוך שיטת main(...) למשל):

```
int children = shmil.numOfChildrenAfterHatching (10, 24);
```

השיטה תחשב תחילה 24% מתוך 10 : כלומר $10 * 24 / 100$ שהם 2.4. לאחר מכן, יומר המספר 2.4 למספר השלם 2, וזה יוצב במשתנה העזר eggsHatched. בשורה הבאה, נחבר את מספר הביצים החדשות שבקעו למספר הצאצאים הקודם של האיגואנה. מספר הצאצאים המעודכן יוחזר בסוף השיטה, ויוצב במשתנה children.

ד. מהלכה למעשה: השיטה-הבונה

עד כה ראינו שני מרכיבים עיקריים בעולם של תכנות מונחה עצמים :

- א. מחלקה – שהיא תיאור מופשט של עצמים מסוג מסוים (למשל המחלקה איגואנה).
- ב. עצמים – שהם המופעים של המחלקה, הבנויים על פי תבניתה (למשל האיגואנות איגי ושמיל). בניגוד למחלקות שהן תבניות בלבד, כל עצם הוא בעל "ישות" של ממש. בתכונותיו מוצבים ערכים המורים על מצבו, וניתן להפנות אליו בקשות להפעיל את שיטותיו.

כיצד ניתן לעבור מרעיון מופשט של מחלקה, אל העצמים ה"ממשיים" איגי ושמיל? כיצד אפשר להשתמש במחלקה כדי לייצר ממנה עצמים?

לשם כך עלינו להשתמש בשיטה מסוג מיוחד, הנקראת **שיטה-הבונה (constructor)**. השיטה-הבונה יוצרת עצמים חדשים, שהם מופעים של המחלקה. שיטה זו שייכת ישירות למחלקה (ולא לעצמים הנוצרים ממנה), והיא חיונית לה. שכן, אחרת לא נוכל לייצר עצמים בתבנית המחלקה, והיא תישאר מנוונת. זו הסיבה שהובילה את יותם להוסיף שיטה-בונה למחלקה, אף שגיון לא ביקש זאת במפורש.

כיצד לכתוב שיטה-בונה?

נראה כיצד יש לכתוב קונסטרקטור, באמצעות ניתוח השיטה-הבונה שכתב יותם במחלקתו:

```
public Iguana (int numOfChildren, double weight, boolean isFriendly){
    this.numOfChildren = numOfChildren;
    this.weight = weight;
    this.isFriendly = isFriendly;
}
```

חתימת הקונסטרקטור

בגיאווה, אין מילה שמורה המורה על כך ששיטה מסוימת היא שיטה-בונה. לעומת זאת, יש מבנה חתימה מיוחד שבעזרתו המהדר מזהה שיטות-בונות. שימו לב כי המבנה המיוחד הוא הדרך היחידה לזהות קונסטרקטורים, ולכן חשוב מאוד להקפיד עליו.

מבנה החתימה של השיטה-הבונה יראה כך:



```
public Iguana (int numOfChildren, double weight, boolean isFriendly)
```

חתימת הקונסטרקטור בנויה כך:

1. שם המחלקה (שמתפקד הן כשם השיטה הן כטיפוס ערך ההחזרה).
2. רשימת הפרמטרים של השיטה בסוגריים עגולים (אם רשימת הפרמטרים ריקה – הסוגריים יהיו ריקים).

וכרגיל, את החתימה מקדימה הרשאת הגישה (על פי רוב **public**).

? חזרו ובדקו את מבנה החתימה של השיטה-הבונה ושל שיטה רגילה. מה מיוחד בחתימת השיטה-הבונה?

מאחורי המבנה המיוחד מסתתר הגיון רב. הרשאת הגישה מסוג *public* חיונית, כדי שמחלקות אחרות יוכלו להשתמש בשיטה הבונה וליצור מופעים של מחלקתנו, כגון איגי או שמיל. ערך ההחזרה של השיטה הוא מופע חדש של המחלקה, ולכן שם הטיפוס של ערך ההחזרה הוא בדיוק כשם המחלקה. כלומר השיטה-הבונה של המחלקה איגואנה מחזירה עצם מטיפוס *Iguana*. שמה של השיטה הוא תמיד כשם המחלקה. כאשר נרצה ליצור עצם חדש נשתמש בקריאה לשיטה הבונה. משום ששם השיטה זהה לשם המחלקה המהדר יוכל לדעת איזה עצם חדש עליו לבנות, ולא יזו מחלקה לפנות כדי למצוא את השיטה-הבונה המתאימה. שם השיטה וערך ההחזרה זההים מאוחדים בחתימה למילה אחת. ולבסוף, כמו בכל שיטה, תיסגר חתימת השיטה בסוגריים עגולים שיכילו את הפרמטרים לשיטה (אם יש כאלה), או יישארו ריקים (אם אין).

גוף השיטה-הבונה

כדי ליצור עצם חדש היכול לתפקד יש לבצע שתי פעולות:

1. להקצות עבור העצם קטע זיכרון מיוחד, שבו יש מקום לכל אחת מתכונות העצם.
2. לאתחל את העצם, כלומר לתת ערכים ראשוניים לתכונותיו.

הפעולה הראשונה תבצע באופן אוטומטי על ידי ג'אווה בכל פעם שנקרא לשיטה הבונה, ואין צורך להוסיפה למימוש השיטה. את הפעולה השנייה ניתן לבצע בדרכים שונות, ולפי הצורך לממש אותה בגוף השיטה-הבונה. שימו לב שאתחול הערכים על ידי המתכנת אינו הכרחי כדי ליצור עצם, אך הוא רצוי, אם ברצוננו שהעצם החדש יוכל לתפקד כהלכה.

נחזור לשיטה הבונה במחלקה איגואנה. כדי לאתחל איגואנה חדשה, צריך לקבוע ערך התחלתי לכל התכונות שלה: משקלה, מספר הצאצאים שלה ורמת החברותיות שלה. במקרה זה, יישלחו הערכים המתאימים על ידי מי שמבקש ליצור איגואנה חדשה, אל הפרמטרים של הקונסטרוקטור בעת הפעלתו.

שימו לב לשמות שבחר יותם לפרמטרים של הקונסטרוקטור. שמות אלו זהים לשמות של תכונות האיגואנה, ולא לשווא. שמות הפרמטרים מייצגים את הערכים המתאימים שיכילו התכונות, ולכן שמות אלו משמעותיים. אולם בגוף השיטה ניתן להבחין בין התכונות עצמן (שאליהן פנינו על ידי שימוש במילה השמורה *this*, בתוספת סימון-הנקודה) ובין הפרמטרים שערכם יוצב בתכונות. כך למשל מוצב הערך של הפרמטר *weight* בתוך התכונה *this.weight*.

שאר השיטה ברור למדי. "פס הייצור" שלנו יאתחל כל אחת מתכונות האיגואנה בהתאם לפרמטר המתאים. כדי להחזיר את ערכי האיגואנה החדשה שנוצרה אין צורך להשתמש במילה *return*. הקונסטרוקטור יחזיר אוטומטית את העצם החדש כנדרש.

בניית עצמים חדשים

ראינו כיצד לכתוב את השיטה-הבונה. נראה עתה כיצד ניתן להשתמש בה בכדי ליצור עצמים חדשים.

כדי ליצור אובייקט חדש ממחלקה מסוימת, נשתמש במילה השמורה *new* באופן הבא:

```
new NameOfClass (...parameters for constructor...)
```

נניח למשל כי ברצוננו ליצור איגואנה חדשה שהיא חסרת ילדים, שמשקלה 3.2 קילו ושהיא זעפנית במיוחד. לשם כך נשתמש במילה השמורה *new*, ובשיטה הבונה של המחלקה איגואנה:

```
new Iguana (0, 3.2, false);
```

השימוש במילה השמורה *new* לפני שם המחלקה מורה לג'אווה על צורך ביצירת עצם חדש מהמחלקה איגואנה. לכן, תיגש ג'אווה אל קוד המחלקה, ותחפש שם את השיטה-הבונה המתאימה. השיטה-הבונה תדאג להקצות באופן אוטומטי זיכרון עבור העצם החדש, וכן תבצע את הפקודות שבגוף השיטה. במקרה זה, תאתחל השיטה את שדות האיגואנה, בהתאם לפרמטרים שנעביר.

חשוב לשים לב כי אין לנו שום מידע על המיקום בזיכרון שבו נוצר העצם החדש.

המחלקה כמגדירה טיפוס

בפסקה הקודמת טענו כי אין לנו כל מידע היכן בזיכרון נמצאת האיגואנה החדשה שיצרנו. כיצד, אם כן, נוכל לפנות אליה? ראינו כי כל מחלקה משמשת כתבנית שניתן ליצור ממנה עצמים. כעת נראה כי למחלקות תפקיד נוסף: כל מחלקה מגדירה טיפוס נתונים.

למשל, לאחר שהגדרנו את המחלקה איגואנה, נוכל להצהיר על משתנים מהטיפוס *Iguana*. במשתנים כאלה נוכל להציב, כמובן, עצמים שנוצרו על ידי השיטה-הבונה של המחלקה איגואנה. עובדה זו מאפשרת לנו נגישות אל האיגואנה החדשה שיצרנו. כל שעלינו לעשות הוא להצהיר מבעוד מועד על משתנה מטיפוס *Iguana*, ולהציב בו את האיגואנה החדשה. למשל:

```
Iguana muki;
```

```
muki = new Iguana (0, 3.2, false);
```

בשורה הראשונה אנו מצהירים על משתנה חדש מטיפוס איגואנה. בפעולת ההצהרה על המשתנה עדיין לא יצרנו שום עצם. כל שעשינו הוא הקצאת משתנה שיוכל להכיל עצם מהמחלקה. בשורה השנייה משתמשים בשיטה הבונה כדי ליצור איגואנה חדשה. לאחר מכן, מוצבת האיגואנה החדשה במשתנה *muki*. לאחר ההצבה, יפנה המשתנה *muki* אל האיגואנה החדשה שיצרנו.

מעניין לשים לב, כי בעוד שההצהרה על המשתנה muki נעשית בשלב ההידור של התוכנית, הבנייה של העצם (כלומר הפעלתה של השיטה-הבונה) הוא תהליך דינמי המתרחש רק בעת ריצת התוכנית.

כרגיל, נוכל לאחד את ההצהרה ואת ההשמה לשורה אחת:

```
Iguana muki = new Iguana (0, 3.2, false);
```

השיטה-הבונה מאפשרת לנו לממש את המחלקה וליצור מהתבנית המופשטת מספר כלשהו של עצמים חדשים.

קונסטרוקטור ברירת מחדל

ראינו כי קונסטרוקטור, בדומה לכל שיטה, יכול לקבל ערכים. לעתים, נרצה לכתוב קונסטרוקטור שאינו מקבל אף ערך. חתימתה של שיטה-בונה כזו תראה כך:

```
public ClassName()
```

מצד אחד, קונסטרוקטור כזה יכול לאתחל את התכונות על פי ערכים הנבחרים בתוך מימוש השיטה-הבונה (שאינם מועברים על ידי המשתמש), אך מצד אחר, הקונסטרוקטור יכול לא לאתחלן כלל (אם כי דרך זו בוודאי אינה רצויה). קונסטרוקטור שאינו מקבל ערכים נקרא בג'אווה **קונסטרוקטור ברירת מחדל (default constructor)**.

כפי שראינו, לשיטה הבונה תפקיד מהותי ביותר בכל מחלקה. רק בעזרתה ניתן להשתמש בהגדרת המחלקה כדי ליצור עצמים ממשיים.

בג'אווה, קיים מנגנון הדואג לכך שבכל מחלקה תופיע שיטה-בונה, גם אם המתכנת שכח להוסיפה. המנגנון מוסיף למחלקה קונסטרוקטור ברירת מחדל באופן אוטומטי אם ורק אם המתכנת לא הגדיר במחלקה אף לא שיטה-בונה אחת. קונסטרוקטור זה אינו מקבל ערכים, אך הוא מאתחל את תכונות העצם לערכי ברירת המחדל המתאימים להם. למעשה, הדבר הראשון שכל שיטה-בונה עושה הוא לאתחל את תכונות העצם לערכי ברירת המחדל, כך שאם המתכנת לא הציב בתכונות ערכים, הן מאותחלות בכל זאת. קונסטרוקטור ברירת מחדל, ככל קונסטרוקטור, מקצה לעצם החדש מקום מיוחד בזיכרון. את השיטה-הבונה שג'אווה הוסיפה אם המתכנת לא הגדיר אף שיטה-בונה, אי-אפשר לראות במפורש בקוד המחלקה, אולם היא מהווה חלק מממשק המחלקה, ומחלקות אחרות יכולות להשתמש בה כדי ליצור עצמים חדשים. על אף האמור לעיל, מומלץ לא לסמוך על קונסטרוקטור ברירת המחדל שמוסיפה ג'אווה, ולהוסיף למחלקה שלכם שיטה-בונה העונה על צרכיכם.

ה. משתנים מכילים עצמים על ידי הפניה

חשוב לדעת כי ההכלה של עצם במשתנה ממומשת באופן מיוחד.
נסתכל על שורות הקוד הבאות:

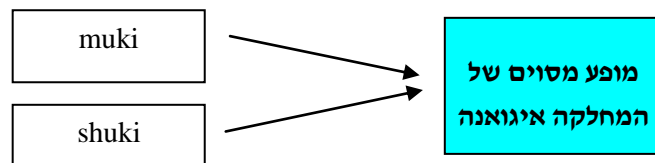
```
Iguana muki = new Iguana (3, 2.2, true);
```

```
Iguana shuki = muki;
```

בשורה הראשונה מוצבת במשתנה מוקי, איגואנה חדשה. בשורה השנייה, מוצבת במשתנה שוקי אותה איגואנה בדיוק.

שימו לב שהאיגואנה "שוקי" אינה רק זהה לאיגואנה "מוקי" בכל תכונותיה, אלא שתיהן אותו העצם ממש! כל שינוי במוקי יגרור אוטומטית שינוי בשוקי, ולהפך. למשל, אם "נאכיל" את העצם המוצב ב-muki (על ידי שימוש בשיטה eat()), נוכל לברר את משקלו החדש, באמצעות העצם המוצב במשתנה shuki. למעשה, "שוקי" ו"מוקי" הם שני כינויים לאיגואנה אחת. כיצד יכול אותו עצם להיות מוצב בשני משתנים בו-זמנית? הוא בוודאי אינו יכול להיות מוצב בשני שטחי זיכרון שונים באותו הזמן!

משתנה המכיל עצם אינו מכיל בשטח הזיכרון שלו את העצם ממש, אלא רק הפניה (reference) אל העצם. במשתנים shuki ו-muki מופיעה הפניה אל אותה איגואנה, וכך יכולים כמה משתנים שונים "להכיל" אותו עצם.



? לו היינו רוצים להציב במשתנה shuki איגואנה אחרת, שלה ערכים זהים לאלה של muki, כיצד היינו עושים זאת?

כפי שנראה בהמשך, העובדה שמשתנים מכילים עצמים על ידי הפניה חשובה, שכן היא תחייב אותנו לפעול בזהירות מסוימת בעת העתקת עצמים והעברתם. אולם חשוב להדגיש כי אופן ההכלה המיוחד הוא שאלה של מימוש, ואינו נוגע לאופן שבו אנו מתייחסים אל עצמים ואל משתנים המכילים אותם. בכל פעם שנפנה אל המשתנה muki, הכוונה היא לעצם המוכל בו. מבחינה זו, משתנים המכילים עצמים אינם שונים ממשתנים המכילים ערכים מטיפוס בסיסי. בשני המקרים השימוש במשתנה מתייחס לערך המוכל במשתנה.

1. שימוש במחלקה

כדי לבדוק שמחלקתו אכן עובדת כראוי, כתב יותם מחלקה נפרדת ובה תוכנית קצרה המשתמשת במחלקה איגואנה. מאחר וכל תפקידה של המחלקה הוא להריץ תוכנית, תכיל המחלקה את השיטה `main(...)`, ולא נוסף לה תכונות או שיטות אחרות.

```
/**
 * This class is a test for the Iguana class, and holds only a main method.
 * During the program shmil eats, and the number of Iguanas on Igi and Shmil's
 * rock after hatching is calculated and printed.
 */
public class TestIguana{
/** A main method to test the Iguana */
    public static void main (String[] args){
        int sum = 0;
        //create 2 new instances of Iguana class:
        Iguana igi = new Iguana (2, 4.2, false);
        Iguana shmil = new Iguana (0, 0, true);
        //add igi and shmil themselves to sum:
        sum += 2;
        //ask user how much seaweed is in the cave:
        int numOfSeaweeds = InputRequestor.requestInt
            ("How much seaweed is in the cave?");
        shmil.eat (numOfSeaweeds);
        // update igi's number of children
        int childOfIgi = igi.numOfChildrenAfterHatching (20, 10.0);
        // calculate igi's and shmil's sum of children
        sum += (shmil.getNumOfChildren() + childOfIgi);
        //print the sum of Iguanas in Igi & Shmil's rock:
        System.out.println ("The number of Iguanas on Igi and Shmil's rock: "
            + sum);
    }
}
```


ז. סיכום

בפרק זה, היכרנו את הכלים הבסיסיים שמשמשים בתכנות מונחה עצמים. ראינו כיצד ניתן להגדיר מחלקות, וכיצד להשתמש בהן כדי ליצור מופעים. עתה, תוכלו לתכנן מחלקות ועצמים, להגדירם כרצונכם ולהשתמש בהם כדי לפתור בעיות.

מושגים

class	מחלקה
attribute	תכונה
method	שיטה
constructor	שיטה-בונה – קונסטרקטור
default constructor	קונסטרקטור ברירת מחדל
object	עצם
reference	הפניה
test program	תוכנית בדיקה

פרק 3 דף עבודה מס' 1

נקודת התחלה

מטרות

1. כתיבת מחלקה פשוטה.
2. בדיקת מחלקה.

מה עליכם לעשות?

בתרגיל זה עליכם לבנות מחלקה בשם Point. מחלקה זו מייצגת נקודה דו-ממדית, שלה קואורדינטות שלמות. כל נקודה מאופיינת על ידי קואורדינטת x וקואורדינטת y. המחלקה תהיה שימושית ביותר לכל תוכנית הקשורה ביישומים גיאומטריים שתכתבו בעתיד.

בתרגיל זה תתנסו גם בבדיקת המחלקה על ידי כתיבת תוכנית בדיקה, שהיא מחלקה שתכלול את השיטה main(...) בלבד.

המחלקה Point

- א. צרו פרויקט חדש.
- ב. בתוך הפרויקט, צרו קובץ חדש בשם Point.java, והגדירו בו מחלקה חדשה בשם Point. (זכרו: שם המחלקה ושם הקובץ שבו היא נמצאת חייבים להיות זהים).
- ג. על המחלקה Point לממש את השיטות הבאות:
 1. שיטה-בונה: מקבלת שני ערכים – קואורדינטת x וקואורדינטת y של נקודה חדשה. שתי הקואורדינטות הן מספרים מטיפוס *int*. השיטה תבנה נקודה חדשה בהתאם לקואורדינטות הנתונות.
 2. השיטה getX(): מחזירה את קואורדינטת ה-x של הנקודה.
 3. השיטה getY(): מחזירה את קואורדינטת ה-y של הנקודה.
 4. השיטה setX(...): מקבלת מספר [ערך] מטיפוס *int*, וקובעת את קואורדינטת ה-x של הנקודה בהתאם.
 5. השיטה setY(...): מקבלת מספר [ערך] מטיפוס *int*, וקובעת את קואורדינטת ה-y של הנקודה בהתאם.
 6. השיטה toString(): מחזירה מחרוזת (String) המייצגת את נתוני הנקודה על פי הפורמט הבא:

(<X> , <Y>)

חתימת השיטה היא:

```
public String toString()
```

- ד. כדי לבדוק שהמחלקה שכתבתם עובדת כראוי, עליכם לכתוב תוכנית בדיקה.

המחלקה TestPoint

הגדירו מחלקה נוספת בשם TestPoint (כמובן, מחלקה זו צריכה להיות מוגדרת בקובץ TestPoint.java).

בתוך המחלקה TestPoint הגדירו שיטת main(...) המבצעת את הפעולות הבאות:

1. בונה נקודה חדשה עם קואורדינטות (7, 43).
2. בונה נקודה חדשה נוספת עם קואורדינטות (5, 5).
3. מדפיסה את שתי הנקודות. את המחרוזת המוחזרת מהשיטה toString() מדפיסה השיטה המוכרת println, באופן הבא:

```
System.out.println (xxx.toString());
```
4. מחליפה בין קואורדינטות ה-x של שתי הנקודות, תוך שימוש בשיטות השונות של המחלקה נקודה.
5. מדפיסה שוב את הנקודות החדשות.

בהצלחה!

פרק 3 דף עבודה מס' 2

תאריך (Date)

מטרות

1. כתיבת מחלקה פשוטה.
2. תרגול תיאורטי של הנושאים הנלמדים בפרק.
3. עבודה עם ממשק.
4. מעקב אחר קוד.

מה עליכם לעשות?

חלק א:

עליכם לכתוב מחלקה בשם Date. מחלקה זו מייצגת תאריך המורכב מיום, מחודש ומשנה. המחלקה צריכה לממש את הממשק הבא, המגדיר אילו שיטות פומביות יהיו למחלקה. התכונות הפרטיות אינן מוזכרות כלל, ועל המתכנת עצמו להחליט אילו תכונות הוא רוצה שיהיו למחלקה.

תיאור השיטה	חתימת השיטה
שיטה-בונה היוצרת עצם מטיפוס Date על פי פרמטרים נתונים	Date (<i>int</i> day, <i>int</i> month, <i>int</i> year)
מחזירה את השנה	<i>int</i> getYear()
מחזירה את החודש	<i>int</i> getMonth()
מחזירה את היום	<i>int</i> getDay()
קובעת את ערך השנה על פי הפרמטר הנתון. ערך הפרמטר הוא מספר שלם לא-שלילי בן ארבע ספרות	<i>void</i> setYear (<i>int</i> yearToSet)
קובעת את ערך החודש על פי הפרמטר הנתון. ערך הפרמטר הוא מספר שלם בין 1 ל-12	<i>void</i> setMonth (<i>int</i> monthToSet)
קובעת את ערך היום על פי הפרמטר הנתון. ערך הפרמטר הוא מספר שלם בין 1 ל-31	<i>void</i> setDay (<i>int</i> dayToSet)
מחזירה <i>true</i> אם ורק אם התאריך קודם לתאריך שהועבר אל השיטה כפרמטר	<i>boolean</i> before (Date other)
מחזירה מחרוזת המייצגת את התאריך בפורמט הבא: <day>/<month>/<year>	String toString()

בצעו את הפעולות הבאות :

1. צרו פרויקט חדש.
2. הגדירו את המחלקה Date, וממשו אותה. ניתן להניח תקינות של כל הקלטים ואין צורך לבצע בדיקות תקינות לגביהם.
3. כתבו תוכנית בדיקה בשם TestDate, ובה בדקו את כל השיטות שכתבתם במחלקה Date, כלומר צרו שני מופעים של Date, והפעילו עליהם את כל השיטות של 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.toString());  
    System.out.println (d2.toString());  
}
```

1) מה יודפס בתום הרצת התוכנית?

2) כמה עצמים מסוג Date נוצרו במחלקה הראשית? הסבירו.

מה צלחה!

פרק 3 דף עבודה מס' 3

שוליית הקוסם

מטרות

1. קריאה של ממשק מחלקה.
2. שימוש במחלקות קיימות.

רקע

לרוע המזל, הצליח שוליית הקוסם להציף את חדר העבודה של הקוסם במים. למרבה המזל, עומדים לרשותו של השוליה שני דליי קסם. הדלי הראשון, בכל פעם שהוא ריק לגמרי, 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.

מה עליכם לעשות?

התוכנית שתכתבו אמורה לפתור את בעייתו של שוליית הקוסם, תוך שימוש במחלקה דלי (אל דאגה, זוהי מחלקה גרפית, שכבר נכתבה עבורכם). התוכנית מורכבת משני שלבים. בשלב הראשון, ניצור הדמיה של הסיפור על ידי בניית החדר ושני דליים בגדלים המתאימים. שלב זה נקרא "מידול הסיפור", שכן אנו מגדירים מודולים שייצגו בתוכנית את הישויות האמיתיות. בשלב השני, תציע התוכנית פתרון לבעייתו של שוליית הקוסם. היא תרוקן את החדר על ידי שימוש בשני הדליים על פי הדרישות המפורטות בהמשך.

אנא קראו את ההוראות עד תומן בטרם תתחילו לעבוד.

הקפידו למלא את דרישות התרגיל באופן מדויק.

מהלך העבודה

1. צרו פרויקט חדש בתיקייה הקיימת Bucket \Chap3 \MyProjects \JCreator\C:.
2. בתרגיל זה תשתמשו במחלקה קיימת בשם Bucket ("דלי"). כדי שתוכלו להשתמש בה עליכם להוסיף אל הפרויקט ספרייה הנקראת BucketLib, שכתבנו עבורכם. ספרייה זו מכילה את המחלקה Bucket ומספר מחלקות נוספות היוצרות ממשק גרפי. ממשק זה יאפשר לכם לבדוק את תוכניתכם בצורה נוחה. כמו כן, זכרו לצרף לפרויקט את קובץ הספרייה io.zip.
3. בתוך הפרויקט הגדירו מחלקה חדשה בשם שתבחרו. בתוך המחלקה הגדירו רק שיטת main(...). התוכנית שתכתבו כדי לפתור את בעיית השוליה תופיע בגוף השיטה (...).main

בניית הסיפור

מידול הבעיה כולל את השלבים הבאים :

א. קליטת הנתונים

כדי לפתור את הבעיה אנו זקוקים למספר 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 נגדיר על ידי קבוע (*final*). ערכו של קבוע זה יהיה 20. את שלושת הנתונים האחרים עליכם לקלוט מהמשתמש על ידי שימוש באובייקט מטיפוס `InputRequestor`. שימו לב, אין צורך לקלוט את כמות המים ההתחלתית בדליים. אנו נבנה את הדליים ריקים מלכתחילה. כמו כן עליכם לבדוק שהקלט שתקבלו עבור כל אחד מהנתונים שתבקשו מהמשתמש אכן תקין. את דרישות התקינות תוכלו למצוא בנספח א', המופיע בהמשך דף עבודה זה. לשם פשטות העבודה, תוכלו לכתוב את התוכנית תחילה ללא בדיקות התקינות, ולהוסיפן בשלב שני.

ב. בניית העצמים

ניצור שלושה עצמים מטיפוס `Bucket`. הדלי הראשון ייצג את החדר (מומלץ ליצור אותו בשם "room"). שני הדליים הנוספים ייצגו את דליי הקסם של שוליית הקוסם. הקפידו לבנות את הדליים בהתאם לקיבולות שהוגדרו בסעיף הקודם. כפי שתוכלו לראות, בממשק של המחלקה `Bucket`, המופיע בהמשך, השיטה-הבונה של מחלקה זו בונה מלכתחילה דליים ריקים. את שני דליי העזר שאירו לעת עתה ריקים. את החדר, לעומת זאת, מלאו בכמות המים ההתחלתית שבחר המשתמש, תוך שימוש בשיטות של המחלקה "דלי".

פתרון הבעיה

כעת תוכלו להפעיל את הדליים שיצרתם כדי לרוקן את המים מהחדר. כזכור, המשימה היא לרוקן את כל המים בחדר. מותר להעביר מים מהחדר רק לדלי הקסם הראשון. מותר לרוקן החוצה רק את דלי הקסם השני. כלומר, הוצאת המים מהחדר תיעשה בשלבים : מים יועברו מהחדר לדלי הראשון, לאחר מכן, הם יועברו מהדלי הראשון לדלי השני, ולבסוף יוצאו מהדלי השני לחצר. העברת המים מהחדר לדלי הראשון אפשרית אך ורק אם הדלי הראשון ריק לחלוטין.

לשם פתרון הבעיה השתמשו בשיטות השונות של המחלקה דלי, כפי שהן מופיעות בממשק המצורף.

הרצת התוכנית

לאחר שהידרתם את התוכנית, תוכלו להריץ אותה כרגיל. 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.

נספח א: דרישות התקינות של הקלט

במהלך דף עבודה זה נדרשתם לבקש מהמשתמש שלושה נתונים: קיבולת שני הדליים וכמות המים ההתחלתית בחדר. 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 (שהוגדרה, כאמור, על ידי קבוע). כמו כן, על כמות זו להיות גדולה או שווה ל-0.
- קיבולתו של הדלי הראשון צריכה להיות קטנה או שווה לגודלו של החדר. כמו כן, קיבולתו של הדלי צריכה להיות גדולה ממש מ-0.
- קיבולתו של הדלי השני צריכה להיות קטנה או שווה לקיבולתו של הדלי הראשון. כמו כן, קיבולתו של הדלי צריכה להיות גדולה ממש מ-0.

ממשק המחלקה Bucket

את ממשק המחלקה ניתן לראות באמצעות דפדפן או באמצעות JCreator (ראו במדריך הטכני בסעיף 4). הממשק מופיע בקובץ Bucket.html הנמצא בתיקייה:

C:\JCreator\MyProjects\Chap3\Bucket

Class Bucket

```
public class Bucket
```

```
extends java.lang.Object
```

This class represents a bucket.

The bucket has a maximum capacity, and a current amount of water

Constructor Summary

Bucket (int capacity, String name)
Constructs a new bucket.

Method Summary

void	empty () empties the bucket completely
void	fill (int amountToFill) fills this bucket with a given amount of liquid.
boolean	isEmpty () checks if this bucket is empty


```
void pourInto(Bucket bucketInto)  
    pours as much content as possible from this bucket into a given bucket
```

Constructor Detail

Bucket

```
public Bucket(int capacity, String name)
```

constructs a new bucket. The new bucket is constructed as empty.

Parameters:

capacity - the maximum capacity of bucket

name - the bucket's name

Method Detail

fill

```
public void fill(int amountToFill)
```

fills this bucket with a given amount of liquid. If this amount is more than bucket's capacity, the bucket is filled as much as possible

Parameters:

amountToFill - the attempted amount to fill

empty

```
public void empty()
```

empties the bucket completely.

isEmpty

```
public boolean isEmpty()
```

checks if this bucket is empty

Returns:

true if and only if the bucket is completely empty

pourInto

```
public void pourInto(Bucket bucketInto)
```

pours as much content as possible from this bucket into a given bucket

Parameters:

bucketInto - the bucket into which we are pouring

מהצחה!

