פרק 4

צלילה לעומק מחלקות ועצמים

יותם פסח על תריסר הודעות פרסומת מייגעות ואפילו על הבדיחה החדשה ששלח לו חברו ניר, ודילג הישר אל הודעת הדוא"ל החדשה ששלח לו ג'ון.

"יותם היקר", נאמר בהודעה, "שמחתי לשמוע שאתה מתקדם עם העבודה על המחלקה איגואנה. בכל אופן, נזכרתי בתוספת: אני זקוק לשיטה שתוכל להשוות את גילן של שתי איגואנות, כלומר שיטה במחלקה איגואנה, שתקבל איגואנה נוספת ותחליט מי מהשתיים מבוגרת יותר. לשם כך, תצטרך להוסיף לכל איגואנה תכונה שתייצג את תאריך הלידה שלה. תוכל להשתמש במחלקה Date, שאותה כבר כתבנו בפרויקט שבו השתתפתי בשנה שעברה (סיפרתי לך עליו, לא?). אני מצרף, אם כך, את הקוד של המחלקה Date ואת הממשק שלה, ואתה יכול להשתמש בהם ואף לשנות אותם אם תצטרך. בהצלחה, ג'ון."

אל ההודעה צורף הממשק הבא של המחלקה Date :

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

א. עצמים מורכבים מעצמים

עד כה טיפלנו במחלקות שבהן התכונות היו מטיפוסים בסיסיים. אולם בפרק הראשון ראינו כי בתכנות מונחה עצמים אפשר לזהות ולבנות עצמים שחלק מתכונותיהם (או כולן) הן עצמים אחרים (למשל מערכת הסטריאו במכונית). עצמים כאלו נקראים **עצמים מורכבים** (composed objects).

במקרה שלפנינו, העצם איגואנה יוכל להיות מורכב, בין השאר, מן העצם "תאריך לידה" של האיגואנה, שהוא מהטיפוס Date, המוגדר על ידי המחלקה Date. לשם כך, עלינו להצהיר במחלקה איגואנה על תכונת תאריך הלידה, באותו אופן שבו הצהרנו על התכונות האחרות:

```
public class Iguana{
    private Date birthday;
    …
}
```

בתחילת כל שיטה-בונה ג׳אווה מאתחלת את התכונות לערכי ברירת המחדל המתאימים. תכונה שאינה מטיפוס בסיסי, אלא מטיפוס המוגדר על ידי מחלקה (כגון Date), מאותחלת לערך ברירת המחדל ***null***. למשל, התכונה birthday המוגדרת למעלה, תאותחל ל-***null***, ופירושו: שהמשתנה אינו מכיל אף עצם.

מובן שלא נרצה להשאיר את התכונה כך שתהיה ריקה מעצם, ובשיטה הבונה שנגדיר בהמשך, נאתחל תכונה זו. אולם לערך ***null*** חשיבות רבה בכתיבת תוכנית בטוחה. באמצעות ערך זה נוכל לבדוק האם תכונה מסוימת מכילה עצם או אינה מכילה. למשל, בטרם נבצע פעולה על התכונה birthday, נבדוק את התנאי הבא:

```
if (this.birthday != null)…
```

לעיתים נזדקק לבדיקה זו לפני שננסה לזמן שיטה של העצם birthday. ניסיון לזמן שיטה או לפנות לתכונה של משתנה, שאינו מכיל למעשה עצם, יגרור שגיאה בזמן ריצת התוכנית.

ב. העמסת שיטות (Overloading)

עתה, משהוספנו למחלקה עוד תכונה, נעדכן את השיטה-הבונה, כך שתאתחל גם תכונה זו.

```
public Iguana (int numOfChildren, double weight, boolean isFriendly){
        this.numOfChildren = numOfChildren;
        this.weight = weight;
        this.isFriendly = isFriendly;
        this.birthday = new Date (1, 1, 2001); //initializing birthday to a default date 1.1.2001
}
```

השיטה-הבונה המעודכנת מאתחלת את התכונה birthday לערך ברירת מחדל 1.1.2001. סביר שנרצה להוסיף לחתימת השיטה-הבונה פרמטר נוסף מטיפוס Date, על מנת שנוכל ליצור

איגואנות שנולדו גם בתאריכים אחרים:

```
/**
 * Another constructor that receives parameters for all attributes
 */
public Iguana (int numOfChildren, double weight, boolean isFriendly, Date birthday){
        …
}
```

מצד אחד, נרצה לשמור את הקונסטרקטור הישן כדי שתוכניות שהשתמשו בו יוכלו להמשיך ולעבוד, ומצד אחר, אנו מעוניינים בקונסטרקטור החדש, בעל ארבעת הפרמטרים. כלומר לפנינו שתי שיטות-בונות בעלות שם זהה. נרצה להיות מסוגלים להשתמש בהן לסירוגין על פי הבעיה שבה אנו דנים – לפעמים יהיה נוח ומתאים יותר להשתמש בשיטה-בונה אחת, ולפעמים – באחרת.

ג'אווה מאפשרת לנו לבצע **העמסה (overloading)** של שיטות. העמסת שיטות, היא האפשרות לקרוא לכמה שיטות שונות באותו שם, ובלבד שחתימותיהן שונות (כלומר, רשימות הפרמטרים שלהן אינן זהות).

מנגנון זה מאפשר לנו לקיים את שתי השיטות-הבונות שבמחלקה איגואנה.

החתימות השונות מאפשרות למהדר לדעת לאיזו שיטה אנו קוראים בכל פעם.

הפקודה:

```
Iguana igi = new Iguana (3, 4.5, false);
```

כתובה על פי מבנה החתימה של השיטה הראשונה. לכן, כאשר ירצה המהדר לבצע פקודה זו, הוא יזהה תחילה את מבנה החתימה, ויפנה אל השיטה-הבונה הראשונה.

אם, לעומת זאת, נקרא לפקודה:

```
Iguana igi = new Iguana (3, 4.5, false, d1);        // d1 הוא עצם מטיפוס תאריך
```

יזהה המהדר את החתימה של השיטה השנייה, ויבנה את האיגואנה איגי לפי מימוש שיטה זו. העמסת שיטות היא כלי שימושי וחשוב.

? כתבו קונסטרקטור ללא פרמטרים המייצר איגואנות ידידותיות, ללא ילדים, שתאריך לידתן 1.1.2001, ומשקלן 50 ק"ג!

העמסת שיטות אינה מוגבלת רק לשיטות-בונות. בפרק הקודם הגדרנו שיטה בשם eat(...) שקיבלה כערך את מספר האצות שאוכלת האיגואנה. נניח כי בארוחה ממוצעת אוכלת איגואנה שתי אצות. פירושו של דבר שפעמים רבות בתוכנית נרצה לקרוא לשיטה עם הערך 2. כדי להקל על המשתמש, נכתוב גרסה נוספת של השיטה (כלומר שיטה נוספת, שאף היא תיקרא eat()). הפעם, לא תקבל השיטה אף ערך, ותעדכן את משקל האיגואנה מתוך הנחה שאכלה בדיוק שתי אצות:

```
/** Simulates the Iguana eating 2 seaweeds */
public void eat() {
  this.weight += 2*WEIGHT_OF_SEAWEED;
}
```

? אנו רוצים לבקש מאיגי לאכול ארבע אצות. הראו שתי דרכים שונות לבצע זאת.

העמסת השיטות מאפשרת לנו לכתוב שיטות רבות שלהן שם זהה, בתנאי שלכל אחת מהן חתימה שונה. זוהי אחת הסיבות שחתימת השיטה נחשבת לחלק מהותי כל כך מהגדרתה.

השיטה eat() והשיטה eat (***int*** numOfSeaweeds) הן שתי שיטות שונות, אף ששמן משותף. את שתי שיטות האכילה יכולנו להגדיר גם ללא העמסת שיטות, אילו היינו קוראים לאחת בשם eat1 ולאחרת בשם eat2. אולם ברור שהגדרה זו מסורבלת ואינה נוחה לשימוש כמו השם הפשוט eat, שבו השתמשנו עבור שתיהן.
בְּמקרה של שיטות-בונות, העמסה היא כלי הכרחי. מפני שבג׳אווה כל שיטה-בונה נקראת תמיד כשם המחלקה, העמסת השיטות היא בעצם הכלי היחיד המאפשר לנו להגדיר כמה שיטות-בונות באותה תוכנית.
הדגמנו העמסת שיטות בעזרת המחלקה המלווה שלנו, האיגואנה. דוגמה זו היא רק מקרה פרטי, ונדגיש שהעמסת שיטות אפשרית בכל מחלקה, גם כשאין מדובר בעצמים מורכבים אלא בעצמים פשוטים.

ג. עותק או הפניה?

כאשר אנו משתמשים בעצם, עולה השאלה האם להשתמש בעצם הזה עצמו, או ליצור העתק שלו, ולהשתמש בהעתק. שאלה זו רלבנטית כשמדובר בשיטות המקבלות עצמים כפרמטרים על מנת לעדכן תכונות לפיהם, בשיטות המאחזרות תכונות, ובשיטות-בונות. בפסקאות הבאות נדון בנושא זה.

פרמטר שהוא עצם

ראינו, אם כן, כי אין קושי להגדיר שיטה-בונה נוספת שתקבל פרמטר נוסף של תאריך לידה.
מה יהיה מימוש שיטה זו? המתכנת התמים עלול להתפתות ולהציב את העצם birthday שבפרמטר ישירות בתכונה birthday.

```
/**
 * A constructor for the class
 * receives parameters with values for all attributes.
 * Builds a new Iguana according to these parameters
 */
public Iguana (int numOfChildren, double weight, boolean isFriendly, Date birthday){
    this.numOfChildren = numOfChildren;
    this.weight = weight;
    this.isFriendly = isFriendly;
    this.birthday = birthday;
}
```

הצבה שכזו תהיה כהכנסת "סוס טרויאני" לתוך האיגואנה, שכן היא תאפשר למשתמש להפעיל את שורת הפקודות הבאה, ובכך לשנות את תאריך הלידה של האיגואנה:

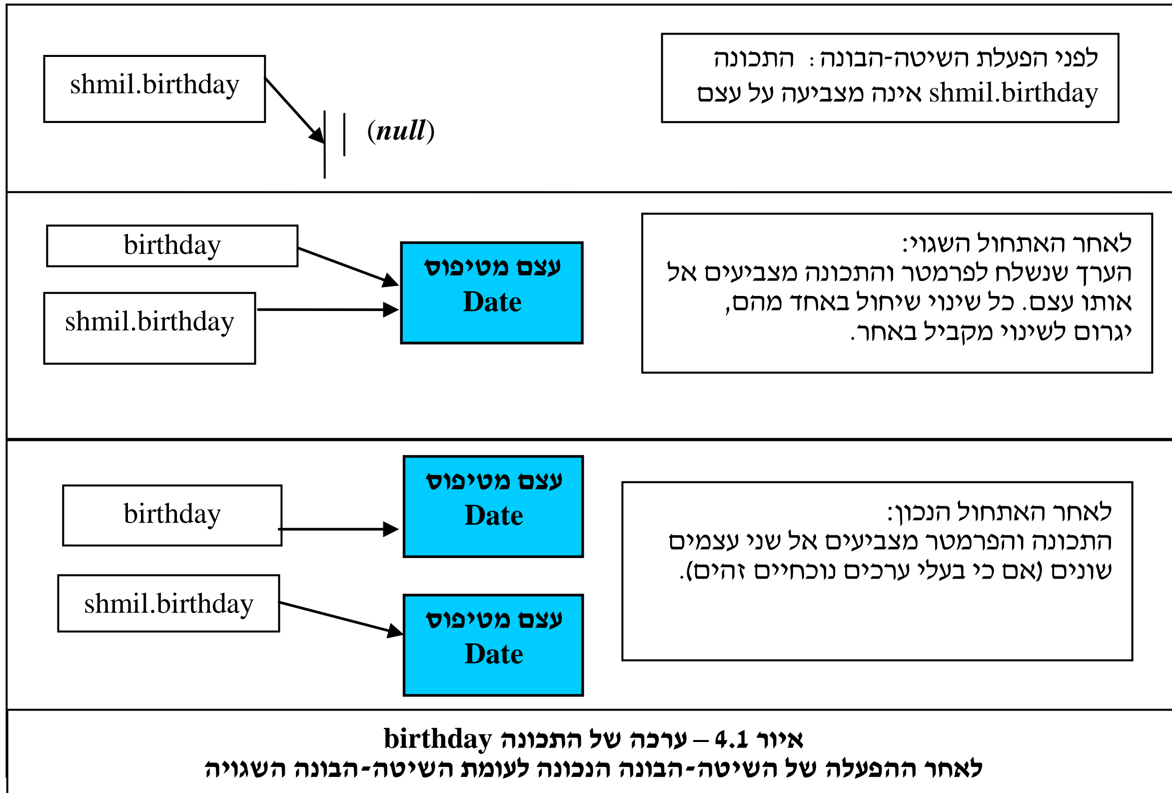
```
Date d1 = new Date (30, 9, 1976);
Iguana shmil = new Iguana (3, 2.7, true, d1);
d1.setYear (1984);
```

? מה יהיה תאריך הלידה של האיגואנה שמיל בסיום הפקודות המוצעות למעלה? מדוע?

לפניכם מימוש נוסף של השיטה-הבונה:

```
/**
* A constructor for the class
* receives parameters with values for all attributes.
* Builds a new Iguana according to these parameters
*/
    public Iguana (int numOfChildren, double weight,
                                                boolean isFriendly, Date birthday){
                …
            //copy birthday  into a new object
            Date copyBirthday = new Date (birthday.getDay(),
                                                birthday.getMonth(), birthday.getYear());
            //set birthday attribute according to copy of the parameter
            this.birthday = copyBirthday;
    }
```

במימוש זה נוצר העתק של תאריך הלידה שקיבלנו מבחוץ, על ידי יצירת עצם חדש, שאותו "ממלאים" בנתוני תאריך הלידה שהתקבל כפרמטר. לאחר מכן, מציבים בתכונה ***this***.birthday הפניה לעצם החדש שיצרנו. בדרך זו נמנעה כניסת "הסוס הטרויאני" שקרתה במימוש הקודם של השיטה-הבונה, שכן התכונה birthday והפרמטר birthday (שניהם מטיפוס Date) מתייחסים כעת לשני עצמים שונים.



איור 4.1 – ערכה של התכונה birthday
לאחר ההפעלה של השיטה-הבונה הנכונה לעומת השיטה-הבונה השגויה

אִחזור עצמים

מקובל להוסיף לתכונות שמסווגות על ידי ***private***, ושלמשתמש יכול להיות עניין בערכן, שיטה המחזירה את ערך התכונה. שיטה זו, תאפשר למשתמש לברר את ערכה של התכונה, אף על פי שהגישה הישירה אליה חסומה. ראינו כבר שיטה כזאת עבור התכונה numOfChildren, שהיא מטיפוס ***int***. כעת נוסיף שיטה דומה עבור התכונה birthday, שהיא עצם מטיפוס Date:

```
public Date getBirthday( ){
  return this.birthday;
}
```

לכאורה המימוש פשוט וברור, אך מתעוררת כאן בעיה.

כדי להבין את הבעיה נסתכל בשורות הקוד הבאות:

```
Date d1 = shmil.getBirthday();
d1.setYear (1984);
```

? איזו בעיה מתעוררת כאן?

כזכור, משתנים רבים יכולים להכיל אותו עצם (שכן ההכלה ממומשת על ידי הפניה). השורה הראשונה מציבה במשתנה d1 את העצם המוחזר מהשיטה getBirthday(). על פי המימוש (השגוי) המוצע למעלה יהיה זה ממש אותו עצם התאריך המוצב בתכונה birthday של שמיל. למעשה, שני המשתנים d1 ו-shmil.birthday יצביעו אל אותו העצם. כעת, כאשר נשנה את העצם d1 בשורה השנייה, תשתנה התכונה birthday של שמיל בהתאם, מבלי שהתכוונו לכך! העובדה ששמיל החזיר את העצם המופיע כתכונה שלו, פגעה ברעיון כימוס התכונות והסתרתן. הרצון שעצמים אחרים יוכלו לברר את ערכי תכונותיו של שמיל, אך לא לשנותם, לא יכול להתממש.

לאחר שאיתרנו את הבעיה, הבה נראה את הפתרון הבא:

```
public Date getBirthday(){
    int day = (this.birthday).getDay();
    int month = (this.birthday).getMonth();
    int year = (this.birthday).getYear();
    Date  copyBirthday = new Date (day, month, year);
    return copyBirthday;
}
```

כלומר במקום להחזיר את העצם birthday עצמו, השיטה getBirthday() יוצרת עצם חדש, העתק בשם copyBirthday, שנבנה בצלמו ובדמותו של העצם shmil.birthday, ומחזירה את ההעתק. כך יכול המשתמש לגלות במדויק את ערכי התכונה, מבלי שתהיה לו אפשרות לשנותה או לפגום אותה.

שימו לב לפנייה המקוננת (***this***.birthday).getDay(). ראשית, יש כאן פנייה לתכונה birthday של

העצם הנוכחי, שהיא עצם מטיפוס Date. שנית, יש פנייה לשיטה ()getDay, של עצם התאריך.

האם יש צורך בהעתקה?

פתחנו את הסעיף (סעיף ג – "עותק או הפניה?") בשאלה: האם – כאשר רוצים להשתמש בעצם כתכונה של עצם אחר – יש להשתמש בעצם עצמו, או ליצור העתק שלו ולהשתמש בהעתק? עד כה ראינו שני מקרים הקשורים לעצם מורכב, שבהם מעדיפים להעתיק עצם ולא להשתמש במקור.

במקרה הראשון העברנו עצם כפרמטר והצבנו אותו בתכונה. דוגמה אחת לכך נמצאת בקונסטרקטור של המחלקה איגואנה, המקבלת את הפרמטר birthday על מנת לאתחל את התכונה birthday לפיו.

במקרה השני אחזרנו תכונה שהיא עצם. דוגמה לכך ראינו בשיטה ()getBirthday, המחזירה את התכונה birthday של האיגואנה. החזרה של התכונה המקורית ולא עותק שלה הייתה חושפת אותה לשינויים חיצוניים.

חשוב לדעת, שהצורך הנוסף בהעתקה אינו קשור בהכרח לעצם מורכב. לעתים נרצה ליצור עותק של עצם כדי שיהיו בידינו שני עצמים זהים. לדוגמה, ישנם משחקים שבהם דמויות מסוגלות לשכפל את עצמן. הדמות המשוכפלת, מרגע שנוצרה, היא דמות עצמאית, שאינה תלויה בדמות המקורית. גם אם הדמות המקורית מושמדת, השיבוט שלה יכול להמשיך להתקיים ולפעול. לכן גם בדוגמה זו נשתמש בהעתקה ולא בהפניה נוספת לעצם המקורי.

מתי לא נשתמש בעותק?

לא בכל הפעמים שנעסוק בעצם מורכב אכן נרצה לבצע העתקה, לעתים נרצה דווקא להשתמש בעצם המקורי. ניקח למשל עצם מטיפוס ילד, השומר כתכונה את ההורה שלו. אם נקבל משתנה מטיפוס הורה כפרמטר של השיטה-הבונה, לא נרצה להעתיק אותו אלא להשתמש בו עצמו. אחרת, לא נוכל ליצור שני ילדים של אותו הורה.

גרוע מכך, העתקת ההורה בשיטה-הבונה של הילד, משמעותה שבכל פעם שילד נולד, ההורה משוכפל. אותה בעיה תתעורר אם ניצור עותק בכל פעם שנבקש מהילד לאחזר את ההורה – מספר הורי הילד יגדל.

נוסף לאלו, אין ליצור עותק של הורה, אלא להשתמש בהורה עצמו מפני ששינוי בהורה צריך לשנות את התכונה 'הורה' של הילד, ולא ליצור הורה חדש המתאים לשינוי.

כדאי לזכור: הפניה נוספת לעצם המקורי היא בעצם מתן שם נוסף לאותו עצם, הנשמר באותו מקום בזיכרון. לעומת זאת, העתקה היא יצירת משהו חדש, עצמאי, התופס מקום נוסף בזיכרון.

אין כללי אצבע, המורים מתי כדאי להעתיק את העצם ומתי להשתמש במקור. הדבר תלוי בהגדרת הבעיה. בכל מקרה לגופו, צריך לקחת בחשבון את ההשלכות של שימוש בהפניה או בעותק.

שיטה-בונה מעתיקה (Copy Constructor)

עד כה הדרך היחידה שראינו להעתקת עצם היא קריאת כל התכונות מהעצם המועתק על ידי השיטות getX(), והעברתן לשיטה-בונה, היוצרת עצם חדש על פי תכונות אלו.

הבה ניזכר במימוש הנכון של השיטה getBirthday() :

```
public Date getBirthday(){
    int day = (this.birthday).getDay();
    int month = (this.birthday).getMonth();
    int year = (this.birthday).getYear();
    Date copyBirthday = new Date (day, month, year);
    return copyBirthday;
}
```

קשיים אחדים מתעוררים בצורה זו של העתקה :

1. אם לעצם יותר משלוש תכונות, הרי שנצטרך לקרוא את ערכה של כל אחת מהתכונות, ולהעבירה לשיטה-הבונה. ייתכן שנצטרך לעשות זאת פעמים רבות במשך התוכנית. צורת העתקה זו עלולה לסבך מאוד את התוכנית.
2. אם חלק מהתכונות פרטיות ואין שיטה לגשת אליהן, לא נוכל לקרוא את ערכי התכונות הללו ולשולחן לשיטה-הבונה.

נעדיף העתקה פשוטה יותר – העתקת העצם כולו, בבת אחת, ולא העתקת כל תכונותיו בזו אחר זו.

על מנת לאפשר זאת חייבים להגדיר במחלקה Date שיטה-בונה נוספת. שיטה-בונה זו תקבל עצם מסוג Date כפרמטר, ותיצור עותק שלו. המימוש שלה יראה כך :

```
public Date(Date date){
    this.day = date.day;
    this.month = date.month;
    this.year = date.year;
}
```

שיטה-בונה כזאת נקראת **שיטה-בונה מעתיקה** (Copy Constructor) והיא שימושית מאוד. היא מאפשרת ליצור העתק של Date בקלות. כך משתמשים בה :

```
public Date getBirthday(){
    Date copyBirthday = new Date (this.birthday);
    return copyBirthday;
}
```

שימו לב שגם כאן בגלל העיקרון של **העמסת שיטות** איננו מוותרים על השיטה-הבונה הקודמת של Date, אלא מוסיפים שיטה-בונה נוספת.

בתחילת הפרק הגדרנו שיטה-בונה של Iguana. גם שיטה זו ניתנת למימוש באופן נכון יותר בעזרת השיטה-הבונה המעתיקה של המחלקה Date. המימוש המלא של השיטה מופיע בסוף הפרק.

ד. תקשורת בין עצמים

אחד העקרונות החשובים בתכנות מונחה עצמים הוא עקרון השימוש בתקשורת בין עצמים לשם ביצוע משימות. השיטה שיותם צריך לכתוב אמורה לבקש מאיגואנה מסוימת לברר האם היא מבוגרת מאיגואנה נתונה. כדי לממש את השיטה, עלינו להשתמש בתקשורת בין העצמים.

? אילו עצמים מעורבים במימוש השיטה?

איגואנה א' מקבלת בקשה לברר האם היא מבוגרת מאיגואנה ב'.
ראשית, היא תפנה לאיגואנה ב' כדי לברר את תאריך ההולדת שלה. נשים לב, כי תאריך הלידה של איגואנה א' אף הוא עצם (מטיפוס Date). מבט קצר בממשק המחלקה Date מגלה כי למופעי המחלקה יש שיטה המשווה שני תאריכים. לכן, תוכל איגואנה א' לפנות אל עצם התאריך (שהוא תכונה שלה), כדי שזה יבדוק איזה משני התאריכים מוקדם יותר.
השיטה המשווה תראה כך:

```
/**
 *  A method that returns true if and only if this
 *  iguana is older than (or same age as) iguana2.
 */
public boolean isOlderThan (Iguana iguana2){
        //find iguana2's birthday, and store it in "birthday2":
        Date birthday2 = iguana2.getBirthday();
        //compare the two birthdays (using class Date's comparing
        //method), and store it in "older".
        boolean older = (this.birthday).before (birthday2);
        //return the result:
        return older;
}
```

נוכל לוותר על משתני הביניים, ולאחד את שורות השיטה כך:

```
public boolean isOlderThan (Iguana iguana2){
  return (this.birthday).before (iguana2.getBirthday());
}
```

? על אילו סוגריים בשורה האחרונה ניתן לוותר? על אילו לא? מדוע?

בסעיף זה ראינו כיצד, על ידי תקשורת בין עצמים, אנו יכולים לחלק משימה מורכבת יחסית, בין העצמים השונים, לפי אופיים: כדי לחשב איזו איגואנה מבוגרת יותר, פונה האיגואנה אל העצמים מטיפוס תאריך,ואלו מבצעים עבורה את החישוב הדרוש.

iguana1.isOlderThan ( )

Method …caller

iguana2.getBirthDay( )

Date2

(***this***.birthday).before(Date2)

Date1

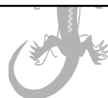
true / ***false***

Date1

true / ***false***

Method …caller

איור 4.2 – תקשורת בין עצמים: שתי האיגואנות והתכונה 'תאריך לידה' של כל אחת מהן.
התוצאה הסופית של התקשורת בין העצמים השונים מחזירה לשואל את תוצאת ההשוואה בין תאריכי הלידה



ה. סיכום

בפרק זה למדנו נושאים מתקדמים בתחום מחלקות ועצמים. למדנו על עצמים המורכבים מעצמים אחרים. בדרך זו נחשפנו לכמה נושאים ששימושם רחב וכללי: העמסת שיטות המאפשרת להגדיר שיטות אחדות בעלות אותו שם, העתקת עצמים, ותקשורת בין עצמים.

מושגים

עצם מורכב	composed object
העמסת שיטות	overloading
שיטה-בונה מעתיקה	copy constructor

```
/**
 * A method that returns the birthday of the Iguana
 * note, that the method returns only a copy of actual attribute
 */
public Date getBirthday(){
    return new Date (this.birthday);
}

/**
* A method that simulates eating two seaweeds.
* updates weight accordingly
*/
public void eat(){
    this.weight + = 2*WEIGHT_OF_SEAWEED;
}

/**
 * A method that receives the number of seaweeds the Iguana eats,
 *  updates weight accordingly
 */
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;
}

/**
 * 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;
}

/**
 * A method that returns true if and only if this Iguana is older than (or same age as) Iguana 2
 */
public boolean isOlder (Iguana iguana2){
      //find Iguana2's birthday, and store in "birthday2":
      //compare the two birthdays (using attribute "birthday"'s comparing method)
      //and store it in "older"
    boolean older = (this.birthday).before (iguana2.getBirthday());
      //return the result:
    return older;
  }
}// end of class
```

השיטה מחזירה ערך שהוא אובייקט

השיטה מקבלת ערך, אך אינה מחזירה ערך

השיטה מקבלת ערכים, ומחזירה ערך

השיטה משתמשת בתקשורת בין עצמים

```java
/**
 * This class simulates an iguana
 * This class 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;
        private Date birthday;
        public final int WEIGHT_OF_SEAWEED = 10;
        // Constructors for class
        /**
         * A constructor for the class
         * receives parameters with values for 3 attributes.
         * builds a new Iguana according to these parameters, and a default value for the last attribute.
         */
    public Iguana (int numOfChildren, double weight, boolean isFriendly){
                this.numOfChildren = numOfChildren;
                this.weight = weight;
                this.isFriendly = isFriendly;
                  //set birthday attribute to a default date 1.1.2001
                this.birthday = new Date (1, 1, 2001);
        }
        /**
         * A constructor for the class
         * receives parameters with values for all attributes.
         * builds a new Iguana according to these parameters
         */
    public Iguana (int numOfChildren, double weight, boolean isFriendly, Date birthday){
                this.numOfChildren = numOfChildren;
                this.weight = weight;
                this.isFriendly = isFriendly;
                  //set birthday attribute according to parameter,
                  //but taking care to copy it into a new object
                this.birthday = new Date (birthday);
        }
        // Regular methods of the class
        /**
         * A method that returns the number of children of the Iguana
         */
        public int  getNumOfChildren(){
              return  this.numOfChildren;
        }
```

הערה כללית המתארת את המחלקה

כותרת המחלקה

הצהרה על תכונות המחלקה

שיטה-בונה של המחלקה

שיטה-בונה נוספת

שימוש בשיטה-בונה מעתיקה

השיטה מחזירה ערך אך אינה מקבלת ערכים

פרק 4 דף עבודה מס' 1

שיטה-בונה מעתיקה (Copy Constructor)

מטרות

תרגול שיטה-בונה מעתיקה (Copy Constructor).

מה עליכם לעשות:

1. צרו פרויקט חדש.
2. הוסיפו אליו את המחלקה Point שהגדרתם בדף עבודה מס' 1 של פרק 3.
3. הוסיפו למחלקה Point את ה-Copy Constructor.
4. לפניכם השיטה הראשית הבאה:

```
public static void main (String[] args){
        Point p1 = new Point (4,9);
        Point p2 = p1;
        Point p3 = new Point (p1);
        p2.setX (5);
        p3.setY (8);
        System.out.println (p1.toString());
        System.out.println (p2.toString());
        System.out.println (p3.toString());
}
```

שאלות

1. שערו מהן תוצאות ההדפסה בלי להריץ את התוכנית? הריצו את התוכנית, ובדקו את תשובותיכם.
2. כמה עצמים מסוג Point נוצרו בתוכנית הראשית הזו? הסבירו.

בהצלחה!

פרק 4 דף עבודה מס׳ 2

ניקח נקודות ונבנה מלבן

מטרות

תרגול עצמים מורכבים.

מה עליכם לעשות?

- הגדירו מחלקה בשם Rectangle, המייצגת מלבן המקביל לצירים.
- כתבו תוכנית בדיקה קצרה שתכלול שיטת (...)main.

הערות

כדי לממש את התרגיל יהיה עליכם להשתמש במחלקה Point המורחבת (הכוללת שיטה-בונה מעתיקה) אשר הגדרתם בדף העבודה הקודם. הוסיפו אותה לפרויקט.

המחלקה Rectangle

המחלקה Rectangle מגדירה מלבן המקביל לצירים.

עליכם לממש את ממשק המחלקה הבא:

תיאור השיטה	חתימת השיטה
שיטה-בונה היוצרת מלבן על פי הפרמטרים הנתונים: הנקודה השמאלית התחתונה, הנקודה הימנית העליונה	Rectangle (Point bottomLeft, Point topRight)
שיטה-בונה היוצרת מלבן על פי הפרמטרים הנתונים: הנקודה השמאלית התחתונה, רוחב המלבן, גובה המלבן	Rectangle (Point bottomLeft, ***int*** width, ***int*** height)
מחזירה את שטח של המלבן	***int*** getArea()
מחזירה את ההיקף של המלבן	***int*** getPerimeter()
מזיזה את המלבן: • על ציר X לפי deltaX: ערך חיובי – הזזה ימינה, ערך שלילי – הזזה שמאלה. • על ציר Y לפי deltaY: ערך חיובי – הזזה למעלה, ערך שלילי – הזזה למטה	***void*** move (***int*** deltaX, ***int*** deltaY)
מחזירה מחרוזת המייצגת את המלבן בפורמט המפורט בהמשך	String toString()

השיטה ()toString תחזיר מחרוזת המייצגת את המלבן בפורמט הבא:

The Rectangle:

Point bottom-left:

(<X > , <Y >)

Point top-right:

(<X > , <Y >)

לדוגמה:

ניצור מלבן על פי הנתונים:

bottomLeft = (2,1);   topRight = (7,5);

לאחר הפעלת השיטה: move(-1, 2), כל הקודקודים ישתנו באותו יחס, והמלבן יזוז שמאלה ולמעלה על מערכת הצירים.

נתוניו החדשים של המלבן יהיו:

bottomLeft = (1,3);   topRight = (6,7);

שימו לב כי בטבלת הממשק לא ציינו את הרשאת הגישה של השיטות. המוסכמה המקובלת היא שכל שיטה המופיעה בממשק היא שיטה המוגדרת בהרשאת הגישה ***public*** (פומבי). בזמן כתיבת ההגדרות של השיטות בקוד המחלקה שלכם, הקפידו להוסיף את הרשאת הגישה לכל חתימה.

המחלקה TestRectangle

מחלקה זו תאפשר לכם לבדוק את המחלקה Rectangle שכתבתם. המחלקה תכלול את השיטה main(...), ובה יהיה עליכם לבצע את הפעולות הבאות:

1. ליצור מלבן על פי נתונים של שני קודקודים.
2. ליצור מלבן נוסף על פי הנתונים (x,y) של הקודקוד השמאלי התחתון, רוחב המלבן וגובהו.
3. להדפיס את מאפייני המלבנים בחלון הפלט הסטנדרטי (console).
4. להזיז את אחד המלבנים למעלה ושמאלה, ואת המלבן האחר ימינה ולמטה.
5. להדפיס את מאפייני המלבנים לאחר התזוזה בחלון הפלט הסטנדרטי.

שאלה:

אילו תכונות בחרתם למחלקה Rectangle? האם ישנה אפשרות נוספת?

הערות:

1. יש ליצור מלבנים עם נתונים תקינים (אין צורך לבצע בדיקות תקינות).
2. הצגת הפלט תיעשה בחלון הפלט הסטנדרטי, כפי שביצעתם זאת בתרגיל הראשון.

בהצלחה!

פרק 4 דף עבודה מס' 3

נוסע ודרכון

מטרות

- תרגול נוסף של יחס ההרכבה.
- תרגול נוסף של שיטה-בונה מעתיקה (Copy Constructor)

מה עליכם לעשות?

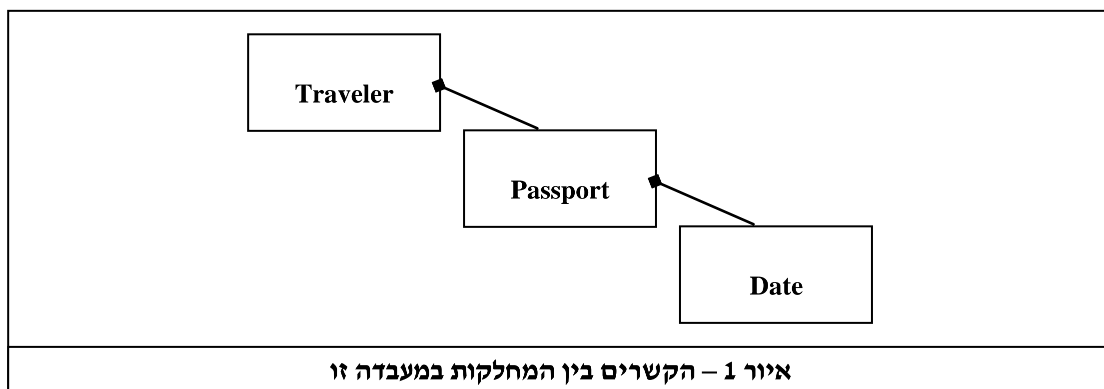
בדף עבודה זה עליכם לממש את המחלקות הבאות:

Passport – מייצגת דרכון של נוסע.

Traveler – מייצגת נוסע כללי.

כדי לממש את המחלקות השתמשו במחלקה Date, שכתבתם בדף עבודה מס' 2 של פרק 3. עליכם להוסיף לה שיטה-בונה מעתיקה (Copy Constructor).

היחסים והקשרים בין המחלקות מתוארים באיור הבא:



איור 1 – הקשרים בין המחלקות במעבדה זו

כזכור, הסימון a ◆—— b מייצג קשר של הרכבה (b מורכב מ- a).

המחלקה Passport

מחלקה זו מייצגת דרכון של נוסע. לשם מימושה, עליכם להשתמש במחלקה Date. עליכם לכתוב אותה כך שתממש את הממשק הבא:

תיאור השיטה	חתימת השיטה
שיטה-בונה היוצרת דרכון חדש על פי הנתונים המועברים אליה: שם, מספר דרכון ותאריך תפוגה	Passport (String name, ***int*** number, Date expiryDate)
שיטה-בונה מעתיקה	Passport (Passport passport)
מחזירה מחרוזת המייצגת את הדרכון (על פי המבנה המתואר למטה)	String toString( )
מחזירה ***true***, אם ורק אם הדרכון תקף בתאריך הנתון	***boolean*** isValid (Date dateChecked)
מעדכנת את תאריך התפוגה של הדרכון	***void*** setExpiryDate (Date newExpiryDate)

המחרוזת המייצגת את הדרכון צריכה להיראות כך:

Name: <name>
Pass. num: <number>
Exp date: <expiry date>

המחלקה Traveler

מחלקה זו מייצגת נוסע. פרטיו של כל נוסע הם הדרכון שלו והמידע האם הנוסע שילם או לא.
שימו לב, על פי היחסים שהוגדרו באיור 1, המחלקה Traveler מורכבת מהמחלקה Passport.

על המחלקה לממש את הממשק הבא:

תיאור השיטה	חתימת השיטה
שיטה-בונה היוצרת נוסע חדש על פי הנתונים	Traveler (Passport passport, ***boolean*** isPayed)
מאפשרת לנוסע לשלם עבור הנסיעה	***void*** pay()
מחזירה ***true***, אם ורק אם הנוסע שילם עבור הנסיעה	***boolean*** isPayed()
מחזירה מחרוזת המייצגת את הנוסע. המחרוזת תהיה זהה לגמרי למחרוזת המייצגת את דרכון הנוסע	String toString( )

מחזירה ***true***, אם ורק אם הנסיעה אפשרית בתאריך הנתון. הנסיעה אפשרית אם דרכון הנוסע תקף בתאריך הנתון, ואם הנוסע שילם	***boolean*** checkTravel (Date travelDate)

בדיקת התוכנית

כדי לבדוק את נכונות התוכנית כתבו מחלקת בדיקה קצרה שתיצור מספר עצמים מטיפוס נוסע, ותדפיס אותם רק אם הנסיעה שלהם אפשרית בתאריך 1 בינואר שנת 2007.

בהצלחה!

פרק 4 דף עבודה מס׳ 4

מיקום (Location)

מטרות

1. תרגול נוסף של הגדרת מחלקה.
2. כתיבת מחלקה שתשמש אתכם בדף העבודה הבא (הצב).
3. תרגול ה-Copy Constructor.

רקע

בדף עבודה זה נגדיר את המחלקה Location (מיקום).

המחלקה Location מייצגת מיקום המוגדר על פי שורות וטורים.

לדוגמה, נסתכל על הלוח הבא:

	1	2	3	4
1		○		
2				
3				X
4				

המיקום של סימן האיקס הוא בשורה מס׳ 3, בטור מס׳ 4, נסמנו כך: (3,4).

המיקום של סימן העיגול הוא בשורה מס׳ 1, בטור מס׳ 2, נסמנו כך: (1,2).

מיקום האיקס, כמו גם מיקום העיגול, הוא עצם מטיפוס Location.

כל עצם מטיפוס Location מיוצג, אם כן, על ידי מספר שורה ומספר טור. אין הגבלה על טווחי המספרים, והם יכולים להיות שליליים.

בדף העבודה הבא נראה שימוש אחד במחלקה ״מיקום״. אולם חשוב להבין כי מחלקה זו היא כללית למדי. לאחר שנגדיר את המחלקה Location, נוכל להשתמש בעצמים מטיפוס ״מיקום״ כדי לייצג נקודות ציון על מפה, משבצות על לוח משחק ועוד.

מה עליכם לעשות?

עליכם להגדיר את המחלקה Location, כך שתממש את כל השיטות המופיעות בממשק הנתון למטה.

ממשק המחלקה Location:

תיאור השיטה	חתימת השיטה
שיטה-בונה היוצרת עצם חדש מטיפוס Location, על פי הפרמטרים הנתונים	Location (***int*** row, ***int*** column)
שיטה-בונה מעתיקה היוצרת עצם חדש מטיפוס Location, כעותק לפרמטר הנתון	Location (Location location)
מחזירה את מספר השורה של מיקום זה	***int*** getRow()
מחזירה את מספר הטור של מיקום זה	***int*** getCol()
מעדכנת את מספר השורה של המיקום, בהתאם למספר הנתון	***void*** setRow (***int*** newRow)
מעדכנת את מספר הטור של המיקום, בהתאם למספר הנתון	***void*** setCol (***int*** newColumn)
מוסיפה למספר השורה הנוכחי את המספר הנתון. המספר הנתון יכול להיות חיובי או שלילי	***void*** addToRow (***int*** numToAdd)
מוסיפה למספר הטור הנוכחי את המספר הנתון. המספר הנתון יכול להיות חיובי או שלילי	***void*** addToCol (***int*** numToAdd)

בהצלחה!

פרק 4 דף עבודה מס׳ 5

צב על אי בודד

מטרות

1. בניית מחלקה על סמך ממשק נתון.
2. שילוב שתי מחלקות כך שעצם ממחלקה אחת ישמש כתכונה למחלקה האחרת.
3. הקפדה על טיפול נכון בהפניות לעצמים והעברתם.

מה עליכם לעשות?

בדף עבודה זה עליכם לכתוב את המחלקה Turtle לפי ממשק למשתמש המוגדר מראש.
תוכנית נתונה שתקבלו תשתמש במחלקה זו. בעזרת התוכנית הנתונה, תוכלו לבדוק האם מימשתם את הממשק במדויק, והאם המחלקה עובדת כנדרש.
אנא קראו את ההוראות עד תומן בטרם תתחילו לעבוד.

לשם כתיבת המחלקות, בצעו את השלבים הבאים:

1. צרו פרויקט חדש בתוך התיקייה הקיימת:

C:\JCreator\MyProjects\Chap4\Turtle

2. שתיים מתכונות הצב (מיקומו הנוכחי ומיקום הבית שלו), ייוצגו על ידי משתנים מטיפוס Location. לשם כך, נצרף לפרויקט את הקובץ Location.java, ובו המחלקה Location, שאותה כתבתם בתרגיל הקודם.
3. צרו בפרויקט מחלקה נוספת בשם Turtle. הגדירו את המחלקה, והצהירו על התכונות שלה. עברו על השיטות שבממשק בזו אחר זו. לכל שיטה, העתיקו במדויק את חתימת השיטה, וממשו אותה. הקפידו לממש את השיטה על פי הדרישות שבממשק (ראו בהמשך). כמו כן, שימו לב לנושא ההפניות, כאשר זה נדרש.

המחלקה צב (Turtle)

תכונותיו של צב:

1. **מיקום באי**. לכל צב מוגדר מיקום באי. האי מחולק למשבצות על פי קווי אורך ורוחב (ראו איור 1 בהמשך). את מיקומה של כל משבצת יש לציין לפי השורה והטור שבהם היא נמצאת. השורות ממוספרות מ-1 בראש המסך ועד אורך האי בתחתית המסך, והטורים ממוספרים מ-1 בשמאל המסך, ועד רוחב האי בימין המסך. שימו לב למשל, כי המשבצת השמאלית העליונה, נמצאת במיקום (1,1) = שורה מס׳ 1, טור מס׳ 1.
צורת מספור זו מקובלת למדי, ונקראת ״סימון בעזרת קואורדינטות גרפיות״.
כדי לשמור את המיקום, עליכם להשתמש באובייקט מטיפוס Location, שאותו הגדרנו בדף העבודה הקודם.
2. **רמת עייפות**. לכל צב רמת עייפות המצוינת באמצעות מספר טבעי בין 0 ל-5. הצב עייף

ביותר כאשר רמת העייפות שלו היא 5.

3. **משבצת בית**. לכל צב יש מיקום אחד (משבצת על הלוח), שבו נמצא ביתו. גם את מיקום הבית תייצגו על ידי עצם מטיפוס Location.
4. **ממדי האי**. כל צב יודע מהו אורכו של האי שבו הוא חי, ומה רוחבו. ידיעת ממדים אלו מאפשרת לצב לא לחרוג ממרחב המחיה שלו.

הצב מתחיל את דרכו במיקום (1,1) ברמת עייפות התחלתית 0. הצב יכול לנוע צפונה, דרומה, מזרחה או מערבה. כל תנועה מעבירה את הצב משבצת אחת לכיוון הנתון. אם הצב מגיע לקצה האי ומתבקש ללכת הלאה, הוא נשאר במקומו. כל צעד מעלה את רמת העייפות של הצב בדרגה אחת.

נניח שמיקומו של צב מסוים הוא (4,2), ורמת העייפות שלו היא 3. אם נבקש מהצב ללכת מערבה, ישתנה מיקומו ל-(4,1), ורמת העייפות שלו תעלה ל-4. אם נבקש ממנו ללכת צעד נוסף מערבה, הוא לא ישנה את מיקומו, וגם רמת העייפות שלו לא תשתנה. אם רמת העייפות של הצב היא 5 והצב מתבקש ללכת לכיוון כלשהו, הוא יתעלם מהבקשה ויישאר במקומו.

הצב יכול לישון, פעולה שמורידה את רמת העייפות שלו ל-0. שיטה נוספת גורמת לצב לחזור ישירות לביתו, פעולה זו תקרא אוטומטית לצב ללכת לישון (ולהוריד את רמת העייפות ל-0). אם הצב עובר בביתו (ללא שימוש בשיטה הישירה) הוא אינו הולך לישון.

קראו את ממשק המחלקה המצורף וודאו שאתם מבינים כל אחת מהשיטות במדויק.

4. בתוך התיקייה C:\Jcreator\MyProjects\Chap4\Turtle, שבה יצרתם את הפרויקט, ישנם קובץ בשם TurtleIsland.java ותיקייה בשם pic שמכילה תמונות. הקובץ TurtleIsland.java מכיל מחלקה ובה תוכנית גרפית שמדמה אי, ומשתמשת במחלקות שאותן מימשתם. **הוסיפו** מחלקה זו לפרויקט, והדרו את הפרויקט כולו. הקובץ הזה מכיל גם את התוכנית הראשית, ולכן לא תצטרכו לממש אותה.

הרצת הפרויקט

הריצו את הפרויקט כרגיל. ההרצה תפעיל את הממשק הגרפי שנכתב עבורכם, ותאפשר לכם לבדוק את המחלקות שאתם כתבתם. על גבי המסך יופיעו כפתורים שונים. לחיצה על הכפתורים למעשה תפעיל את השיטות שאתם כתבתם במחלקות.

איור 1[1]

הצב נמצא כעת במיקום (3,5): שורה מס׳ 3, טור מס׳ 5.

[1] מפת האי צוירה בעזרת העורך של המשחק Age Of Empires של חברת Microsoft.

ממשק המחלקה Turtle

שימו לב: עד כה, בממשקים שראינו הופיעו שיטות בלבד. זו הפעם הראשונה שתכונה מופיעה בממשק, שכן זו תכונה פומבית. תכונה מקבלת הרשאת גישה ***public*** כאשר היא תכונה בעלת ערך קבוע (***final***), שאינו ניתן לשינוי. מכיוון שערך התכונה אינו ניתן לשינוי, אין סכנה במתן הרשאת גישה פומבית. כמוסכמה, תכונה כזו כותבים באותיות גדולות.

תכונה המציינת את רמת העייפות המקסימלית של צב. כאשר צב מגיע לרמה זו הוא אינו יכול לזוז, וחייב לישון	***final int*** TIRED
שיטה-בונה היוצרת צב. הפרמטרים: אורך ורוחב של האי, המיקום של הבית. המיקום ההתחלתי של הצב הוא (1,1), העייפות ההתחלתית היא 0	Turtle (***int*** islandLength, ***int*** islandWidth, Location home)
מחזירה את מיקום הבית של הצב	Location getHome()
מחזירה את רמת העייפות של הצב	***int*** getLevelOfTired()
מחזירה את המיקום של הצב	Location getLocation()
שולחת את הצב הביתה, בכל רמת עייפות. כאשר הצב מגיע הביתה הוא הולך לישון **אוטומטית**, ורמת העייפות מתאפסת	***void*** goHome()
מזיזה את הצב מיקום אחד מזרחה (ימינה)	***void*** goEast()
מזיזה את הצב מיקום אחד מערבה (שמאלה)	***void*** goWest()
מזיזה את הצב מיקום אחד צפונה (למעלה)	***void*** goNorth()
מזיזה את הצב מיקום אחד דרומה (למטה)	***void*** goSouth()
גורמת לצב ללכת לישון. רמת העייפות מתאפסת	***void*** sleep()

בהצלחה!

פרק 4 דף עבודה מס' 6

תרגיל תיאורטי

מטרות

תרגול תיאורטי של הנושאים שנלמדו בפרק.

מה עליכם לעשות?

1. השלימו את חתימות השיטות של המחלקה A :

ממשק המחלקה A :

שיטה-בונה היוצרת עצם חדש לפי הפרמטר : ***double*** x	
שיטה-בונה היוצרת עצם חדש ללא פרמטרים, x מאותחל לאפס	
שיטה-בונה היוצרת עותק של הפרמטר : A a	
מחזירה את הערך של x	
משנה את הערך של x לפי הפרמטר	
מחזירה מחרוזת המייצגת את A בפורמט הבא : A : <x>	

2. ממשו את המחלקה A.
3. מהן התכונות החייבות להופיע במחלקה A? נמקו את תשובותיכם.
4. מהו מספר השיטות-הבונות של המחלקה? כיצד נקרא המנגנון שמאפשר יותר משיטה-בונה אחת?
5. לפניכם תוכנית ראשית המשתמשת במחלקה A :

```
public static void main (String[] args){
        A a1 = new A (7);
        A a2 = new A (a1);
        A a3 = a2;
        a1.setX (10);
        a3.setX (5);
        System.out.println (a1.toString());
        System.out.println (a2.toString());
        System.out.println (a3.toString());
}
```

בתוכנית זו השתמשנו בשם setX (***int*** x) עבור השיטה המשנה את ערך x. מותר לכם להשתמש בשמות אחרים, לפי הממשק שהשלמתם בסעיף 1.

* מה יודפס בתום הרצת התוכנית.
* כמה עצמים מסוג A נוצרו במחלקה הראשית? הסבירו.

6. לפניכם קוד המחלקה B :

```
public class B{
    private A a;
    public B (A a){
        this.a = a;
    }
    public A getA(){
        return this.a;
    }
    public String toString(){
        return "B:" + a.toString ();
    }
}
```

בתוכנית הראשית קיימת השיטה הבאה :

```
public static void main (String[] args){
    A a1 = new A (10);
    B b = new  B(a1);
    a1.setX (7);
    A a2 = b.getA();
    a2.setX (13);
    System.out.println (a1.toString());
    System.out.println (a2.toString());
    System.out.println (b.toString());
}
```

מה תהיה תוצאת ההדפסה?

7. תקנו את הקוד של המחלקה B כך שההדפסה של התוכנית הראשית שלמעלה תהיה :

```
A: 7
A: 13
B: A: 10
```

8. הסבירו בקצרה מהן הבעיות הקיימות כאשר שתי הפניות פונות לאותו מקום בזיכרון.

בהצלחה!