

פרק 7

מחלקות מופשטות

בפרק הקודם הכרנו את אחד הרעיונות המרכזיים בתכנות מונחה עצמים: הפולימורפיזם – רב-צורתיות. הפולימורפיזם הוא האפשרות להתייחס לאובייקטים ממחלקות שונות, בעזרת טיפוס משותף, המשקף צורת התנהגות משותפת. מימוש התנהגות זו יכול להשתנות ממחלקה למחלקה. בפרק הקודם הכרנו את מנגנון הירושה המאפשר פולימורפיזם. בשני הפרקים הבאים נכיר עוד שני מנגנונים חשובים בג'אווה: **מחלקות מופשטות (abstract classes)** ו**ממשקים (interfaces)**. כפי שנראה, מנגנונים אלו מרחיבים את אפשרויות המימוש של רעיון הפולימורפיזם.

עד כה, פגשנו מחלקות רבות. מטרתן העיקרית הייתה להגדיר תבנית שנוכל ליצור עצמים לפיה. בנוסף, ראינו כי כל מחלקה מגדירה טיפוס נתונים חדש, וכן כיצד ניתן להשתמש במחלקה כדי ליצור ממנה תת-מחלקות, תוך שימוש במנגנון הירושה. בפרק זה נכיר מחלקות המשמשות הן כדי להגדיר טיפוס והן כמחלקות-על במנגנון הירושה, אף שלא ניתן ליצור מהן עצמים.

א. מחלקות מופשטות: על שום מה?

נניח לדוגמה שאנו כותבים תוכנת ציור (בסגנון Paint Brush). בתהליך התכנון החלטנו שאחת המחלקות המרכזיות שצריכה להופיע בתוכנית היא "צורה גיאומטרית", ובשלב שני אנו פונים למדל אותה. במבט ראשון, המשימה נראית פשוטה: לצורה גיאומטרית יש תכונות (למשל היקף, שטח, צבע); וגם שיטות (למשל "ציירי את עצמך()", "חשבי את שטחך()").

אולם במבט שני, נגלה שהצורה הגיאומטרית מופשטת מכדי שנוכל לממש אותה במלואה. כיצד תוכל צורה גיאומטרית כללית לצייר את עצמה? הרי אופן הציור יהיה שונה לגמרי אם מדובר במעגל או במלבן! גם התכונות הנדרשות כדי לאפיין כל צורה הן שונות: למלבן תכונות אורך ורוחב, ואילו למעגל – תכונת רדיוס. תכונות אלו דרושות כדי לממש את השיטות "ציירי את עצמך()" ו-"חשבי את שטחך()" של המחלקה צורה גיאומטרית.

העובדה שלא נוכל ליצור עצמים מטיפוס "צורה גיאומטרית" מובילה אותנו למסקנה הבאה: הצורה הגיאומטרית אכן מייצגת מושג או רעיון (שכן יש לצורות גיאומטריות מאפיינים והתנהגות משותפים), אבל זהו רעיון מופשט מכדי שנוכל לממש אותו במלואו בשלב זה. משום שאיננו יכולים לממש את המחלקה במלואה, הרי שגם לא נרצה ליצור ממנה עצמים: עצמים כאלה לא יהיו ממומשים במלואם, ולכן לא יוכלו לתפקד כנדרש. אף-על-פי-כן, נוכל ליצור תת-מחלקות מפורטות יותר היורשות מהמחלקה "צורה", ומהן ליצור עצמים.

* לשם הקיצור, מכאן ואילך נשתמש במונח "צורה" בכל פעם שנתכוון ל"צורה גיאומטרית".

בג'אווה קיים מנגנון מיוחד המאפשר ליצור מחלקות המייצגות רעיונות מופשטים, כדוגמת המחלקה "צורה". מנגנון זה מאפשר להגדיר מחלקה מופשטת (abstract class), ואותה איננו חייבים לממש במלואה, ואף לא ניתן ליצור ממנה מופעים.

ב. הכרזה על מחלקות מופשטות

כדי להגדיר מחלקה מופשטת, נוסיף לכותרת המחלקה את המילה השמורה *abstract* באופן הבא:

```
/** An abstract class defining the abstract concept of a geometric shape*/  
public abstract class Shape {  
    ...  
}
```

שימו לב, אם המחלקה הוגדרה כמופשטת, לא ניתן ליצור ממנה עצמים. לו היינו מנסים לקרוא לפקודה הבאה:

```
 Shape shape1 = new Shape(); //Illegal!
```

היינו מקבלים מהמהדר הודעת שגיאה המסבירה כי לא ניתן ליצור מופעים ממחלקה מופשטת:
Error: class Shape is abstract; cannot be instantiated.

ג. חלקים מהמחלקה המופשטת ניתן לממש כרגיל

אם במחלקה המופשטת ישנן תכונות או שיטות רגילות (הניתנות למימוש), נוסיף אותן לקוד כרגיל:

```
/**An abstract class defining the abstract concept of a geometric shape*/  
public abstract class Shape {  
    // attributes:  
    /** represents the color of this shape */  
    protected Color color;  
  
    // methods:  
    /** returns the color of this shape*/  
    public Color getColor(){  
        return this.color;  
    }  
}
```

שימו לב לשתי עובדות מעניינות:

בפרקים הקודמים הקפדנו תמיד להוסיף למחלקות שיטה-בונה אחת לפחות, ולא הסתמכנו על קונסטרוקטור ברירת המחדל שמוסיפה ג'אווה. לעומת זאת, במחלקה המופשטת "צורה" בחרנו שלא להוסיף שיטה-בונה. מובן שג'אווה תוסיף בעצמה קונסטרוקטור ברירת מחדל, כמו במחלקה רגילה שנכתבה ללא שיטה-בונה. הואיל והמבחין העיקרי בין מחלקות מוחשיות למחלקות מופשטות הוא העובדה שלא ניתן ליצור מהמחלקות המופשטות עצמים, ההקפדה על הוספת שיטה-בונה חשובה פחות. עם זאת, חשוב לשים לב, כי אף שלעולם לא ניתן ליצור עצמים

ממחלקה מופשטת, נוכל (ולעתים אף נרצה) להוסיף למחלקה מופשטת שיטות-בונות, כפי שנראה בהמשך.

בנוסף, שימו לב להרשאת הגישה של התכונה `color`. מכיוון שהמחלקה המופשטת שכתבנו תשמש כמחלקת-על למחלקות שירשו ממנה בעתיד, חשוב שלתת-המחלקות תהיה גישה לתכונה `color`. גישה כזו מושגת על-ידי הרשאת הגישה `protected`.

ד. הגדרת שיטות מופשטות

עד כה, הוספנו למחלקה `Shape` תכונה רגילה ושיטה רגילה, שאינן מצדיקות כשלעצמן את הגדרתה כמחלקה מופשטת. אולם כעת, נרצה להוסיף למחלקה את השיטה `getArea()`, המבקשת מצורה להחזיר את שטחה. שיטה זו מתאימה למחלקה `Shape` מאחר שהיא משותפת לכל הצורות, אולם כאמור, לא נוכל לממש אותה מבלי שנדע לאיזו צורה אנו נדרשים.

אמנם לא ניתן לממש את השיטה, אך ניתן להגדיר אותה כשיטה מופשטת – `abstract method`, המוגדרת על ידי חתימה בלבד, בלי שנכתב בעבורה מימוש. שיטה כזו מייצגת התנהגות המשותפת לכל תת-המחלקות שיירשו מן המחלקה, אך אינה ניתנת למימוש ברמה המופשטת. כפי שנראה בהמשך, מימוש השיטה יעשה ברמת תת-המחלקות.

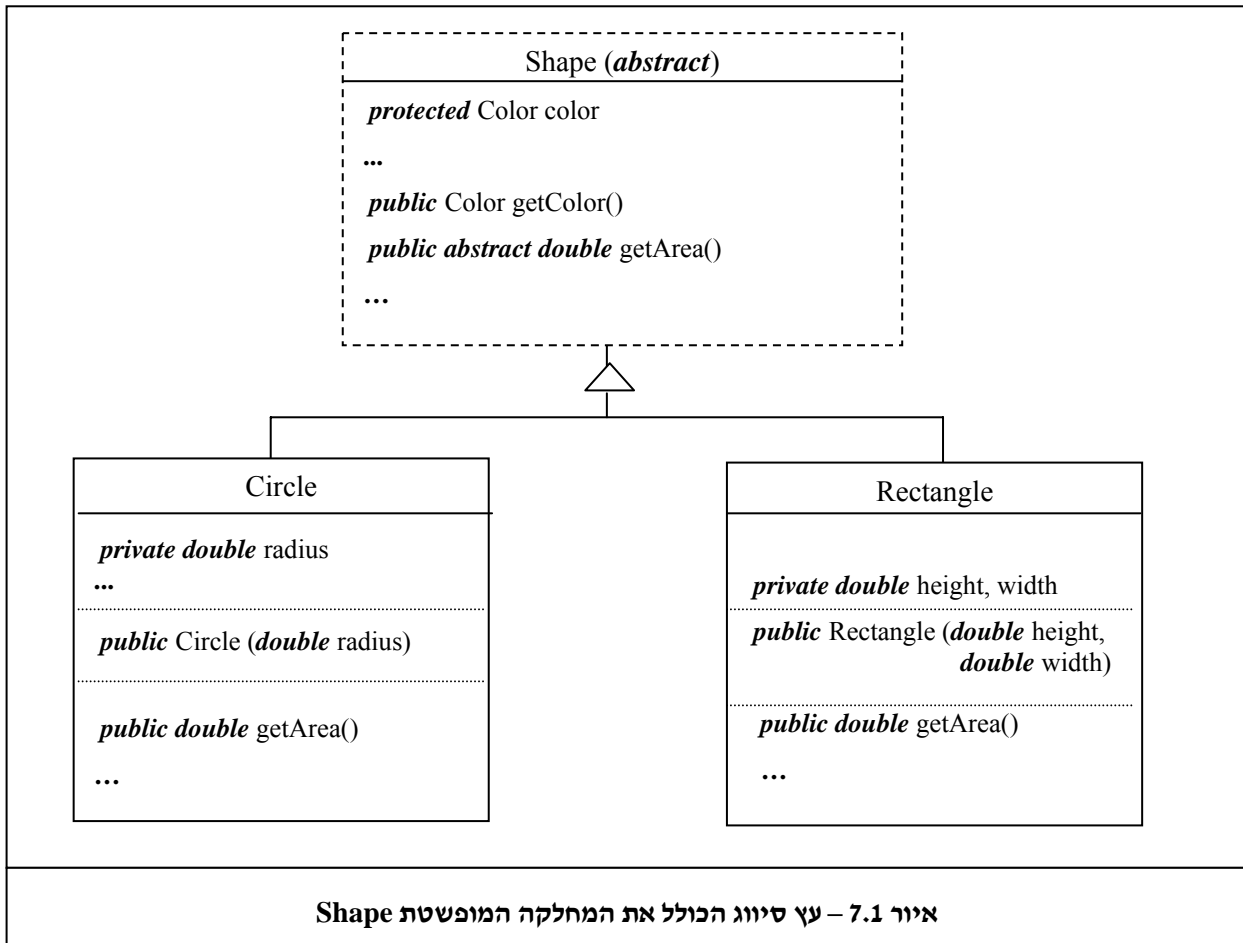
כדי להגדיר שיטה מופשטת בג'אווה נוסיף לחתימת השיטה את המילה השמורה `abstract`. את החתימה נחתום בנקודה-פסיק, ללא סוגריים מסולסלים, וכמובן, ללא מימוש. לדוגמה, נוסיף לקוד המחלקה `Shape` את השיטה המופשטת הבאה:

```
public abstract double getArea();
```

? כיצד תממשו את השיטה `getArea()` של המחלקה `Circle`?

ה. מימוש השיטות המופשטות על ידי הגדרתן מחדש (Overriding)

התבוננו בעץ הסיווג המתואר באיור 7.1: המחלקה "מלבן" יורשת מהמחלקה המופשטת "צורה". במחלקה "צורה" מוגדרת השיטה `getArea()`, כשיטה מופשטת. בעבור המחלקה מלבן לעומת זאת, אנו יודעים לחשב את שטחו של המלבן על ידי הכפלת גובהו ברוחבו. לכן, איננו צריכים להשאיר את השיטה כמופשטת, ואנו יכולים לכתוב לה מימוש. לשם כך, הגדרנו במחלקה "מלבן" שיטה משל עצמה בשם `getArea()`, עם חתימה מתאימה. למעשה, המחלקה "מלבן" מגדירה מחדש (`overrides`) את השיטה המופשטת, ומממשת אותה.



חתימת השיטה במחלקה היורשת זהה לזו שבמחלקת-העל, פרט להשמטת המלה השמורה *abstract* מחתימת השיטה במחלקה היורשת. כך בדוגמה שבאיור לעיל: במחלקת-העל חתימת השיטה היא:

```
public abstract double getArea()
```

ובתת-המחלקות המגדירות את השיטה מחדש, החתימה היא:

```
public double getArea()
```

1. מחלקות מופשטות יוצרות חוזה

הגדרת מחלקה כמופשטת יוצרת מעין חוזה בין המחלקה המופשטת לתת-המחלקות שיוורשות ממנה. כמו בכל חוזה, למחלקה היורשת ממחלקה מופשטת יש "חובות" שעליה למלא, ו"זכויות", כלומר יתרונות מכך שהיא יורשת ממחלקה מסוימת.

חובות המחלקה היורשת

כל מחלקה שרוצה לרשת ממחלקה מופשטת, מתחייבת לממש את כל השיטות המופשטות, או להישאר מחלקה מופשטת בעצמה. אם, למשל, לא נממש את השיטה המופשטת `getArea()` במחלקה "מלבן", המהדר יחייב אותנו להכריז על המחלקה "מלבן" כמחלקה מופשטת, וגם ממנה לא נוכל ליצור עצמים.

העובדה שהמחלקה `Rectangle` יורשת מן המחלקה `Shape` מעמידה בפנינו שתי אפשרויות פעולה:

1. המחלקה `Rectangle` תממש את כל השיטות המופשטות של המחלקה `Shape` – כך תמלא `Rectangle` את חלקה בחוזה ותהפוך למחלקה רגילה הממומשת במלואה. זכרו, ממחלקה כזו ניתן ליצור עצמים.
2. המחלקה `Rectangle` תממש רק חלק מהשיטות המופשטות, או אף אחת מהן, ותכריז על עצמה כמחלקה מופשטת – כך תודיע המחלקה שהיא אינה ממלאת עדיין את התחייבות המימוש, ותשאיר זאת למחלקות שנמצאות מתחתיה בעץ הירושה. כתוצאה מכך, גם מהמחלקה "מלבן" לא נוכל ליצור עצמים.

בבואנו לממש את המחלקה "מלבן", לא סביר שנבחר באפשרות הפעולה השנייה, כלומר – לא נותיר חלק מן השיטות ללא מימוש, ולפיכך לא נגדיר את המחלקה כמופשטת. אולם, נניח כי נרצה להוסיף לעץ הירושה את המחלקה "מצולע" (`Polygon`). מחלקה זו תהיה תת-מחלקה של המחלקה "צורה", ומחלקת-על של המחלקות "מלבן", "משולש" ו"טרפז" (אך לא מחלקת-על של "מעגל"). המחלקה "מצולע" תכיל מאפיינים נוספים על אלו של המחלקה "צורה" (למשל התכונה "מספר הצלעות במצולע"), אך עדיין תהיה מופשטת מכדי לממש את השיטה "צייר את עצמך()". לכן נגדיר גם את "מצולע" כמחלקה מופשטת, ונממש את השיטות המופשטות רק בתת-המחלקות, כגון "מלבן".

שימו לב, כי כל מחלקה שמכילה שיטה מופשטת **חייבת** להיות מוגדרת כמופשטת. אם הוגדרה שיטה מסוימת במחלקה כ-`abstract`, יחייב המהדר להגדיר את המחלקה כולה כ-`abstract`.

זכויות המחלקה היורשת

זכויות המחלקה היורשת אינן שונות מזכויותיה של כל מחלקה המפרטת מחלקת-על. לדוגמה, כאשר המחלקה "מלבן" יורשת מן המחלקה "צורה", היא מקבלת אוטומטית את כל איברי המחלקה "צורה". פירושו של דבר, שלכל מלבן מוגדרות תכונות המחלקה "צורה" (למשל תכונת הצבע) ושיטות המחלקה (למשל "החזירי את צבעך()"). אם השיטות כבר ממומשות במחלקת-העל המופשטת, ניתן להשתמש בהן כפי שהן או להגדירן מחדש בתת-המחלקה. אם הן שיטות

מופשטות, חובה להגדיר אותן מחדש (או להשאיר את תת-המחלקה כמופשטת). נוסף על כך, כמו בכל שימוש בירושה, נוכל להציב עצמים מתת-המחלקה במשתנים מטיפוס מחלקת-העל – כך נוכל לממש את רעיון הפולימורפיזם, כפי שנראה בהמשך הפרק.

ז. סיכום ביניים

- מחלקה מופשטת היא מחלקה שמייצגת רעיון מופשט.
- כדי להגדיר מחלקה מופשטת בג'אווה יש להוסיף בכותרת המחלקה את המילה השמורה *abstract*.
- לא ניתן ליצור עצמים ממחלקה מופשטת, אבל ניתן להגדיר תת-מחלקות שיורשות ממנה.
- מחלקה מופשטת יכולה להכיל שיטות מופשטות, שהן הצהרה על שיטות ללא מימוש.
- כדי להגדיר שיטה מופשטת בג'אווה יש להשתמש במילה השמורה *abstract* בחתימת השיטה, ולהכריז עליה מבלי לממש אותה.
- השיטות המופשטות הן הכלי העיקרי שמספקות המחלקות המופשטות. הן יוצרות ממשק משותף, ואותו כל תת-המחלקות היורשות, שאינן מופשטות בעצמן, צריכות לממש.
- מחלקות מופשטות מגדירות חוזה: על כל תת-מחלקה לממש את כל השיטות המופשטות, או להישאר מחלקה מופשטת בעצמה. תת-המחלקה זוכה לכל היתרונות שירושה מקנה.

הגדרה לדוגמה

בסעיף זה נראה הגדרה של המחלקה המופשטת Shape ושל שתי מחלקות מוחשיות היורשות ממנה: Circle ו-Rectangle. המחלקה Shape מכילה שיטה רגילה ושתי שיטות מופשטות. את תת-המחלקות Circle ו-Rectangle לא הבאנו במלואן. אף-על-פי-כן, שימו לב כי הקפדנו להגדיר שיטה-בונה לכל אחת מהמחלקות. כמו כן, כל אחת משתי המחלקות מממשת את השיטות המופשטות של המחלקה על ידי הגדרתן מחדש (חתימה עם מימוש). שימו לב במיוחד לאופן המימוש השונה של השיטה `getArea()` בשתי המחלקות.

```
/** An abstract class defining the abstract concept of a geometric shape */
public abstract class Shape{

    /** Represents the color of this shape */
    protected Color color;

    // Implemented methods
    /** Returns the color of this shape */
    public Color getColor(){
        return this.color;
    }
}
```

```

// Abstract methods
/** Draws this shape on the screen */
public abstract void draw();

/** Returns the area of this shape */
public abstract double getArea();
}

```

```

/** This class represents a rectangle */
public class Rectangle extends Shape{

    private double height;
    private double width;

    /** Constructs a rectangle with a given height and width */
    public Rectangle (double height, double width){...}

    /** Draws this rectangle.
     * Overrides abstract method draw() in class Shape */
    public void draw(){...}

    /** Returns the area of this rectangle
     * Overrides abstract method getArea() in class Shape */
    public double getArea(){
        return (this.height * this.width);
    }
}

```

```

/** This class represents a circle */
public class Circle extends Shape{

    private double radius;

    /** Constructs a circle with a given radius */
    public Circle (double radius){
        ...
    }

    /** Draws this circle.
     * Overrides abstract method draw() in class Shape */
    public void draw(){...}

    /** Returns the area of this circle
     * Overrides abstract method getArea() in Shape */
    public double getArea(){
        //calculates area by  $PI * r^2$  and returns it:
        return (Math.PI * radius * radius);
    }
}

```


ח. מחלקות מופשטות ושיטות-בונות

בסעיפים הקודמים ראינו כי מחלקה מופשטת היא מחלקה שלא ניתן ליצור ממנה עצמים. אף-על-פי-כן, ניתן להוסיף למחלקות מופשטות שיטה-בונה (אחת או יותר), המאתחלת את התכונות לפי הצורך. כמובן, לא נוכל ליצור עצמים ישירות על ידי שיטה-בונה של מחלקה מופשטת. השימוש בשיטה-הבונה יהיה שימוש עקיף בעזרת השיטות-הבונות של תת-המחלקות, כלומר על ידי מנגנון *super*.

נניח כי אנו רוצים לוודא שלכל צורה תאוחל תכונת הצבע שלה. לשם כך, נוכל להוסיף למחלקה "צורה" את השיטה-הבונה הבאה:

```
public Shape (Color myColor){  
    this.color = myColor;  
}
```

אף-על-פי שהוספנו שיטה-בונה, לא נוכל ליצור בעזרתה עצמים באופן ישיר. קריאה לפקודה הבאה תגרור שגיאת הידור:

```
 Shape myShape = new Shape (Color.red); //ILLEGAL!
```

אף-על-פי-כן, בשיטה-הבונה שכתבנו נוכל להשתמש בתוך השיטות-הבונות של תת-המחלקות. למשל, נוסיף למחלקה "מעגל" שיטה-בונה המשתמשת בשיטה שהוספנו למחלקה "צורה":

```
public Circle (double radius, Color myColor) {  
    super (myColor);  
    this.radius = radius;  
}
```

קריאה לשיטה זו תבנה מעגל: צבעו יאוחלל בעזרת-השיטה הבונה המתאימה במחלקה "צורה" והרדיוס שלו יאוחלל ישירות בתוך השיטה-הבונה של המחלקה "מעגל". שיטה-בונה זו מבטיחה כי כאשר נקרא לשיטה-הבונה של המחלקה "מעגל", יאוחלל גם צבעו של המעגל (שהוא בפרט – צורה). למחלקה "צורה" נוכל להוסיף גם שיטה-בונה אחרת שאינה מקבלת ערכים ומאתחלת את תכונת הצבע באופן קבוע:

```
public Shape(){  
    this.color = Color.red;  
}
```

שיטה זו מאתחלת את תכונת הצבע של הצורה על ידי התכונה הפומבית red השייכת למחלקה בשם Color (ההנחה היא שתכונה זו מייצגת את הצבע האדום). גם במקרה זה נוכל להשתמש בשיטה-הבונה בתת-מחלקה. למשל במחלקה "מלבן" נוכל להוסיף את השיטה-הבונה הבאה:

```
public Rectangle (double height, double width){  
    super();  
    this.height = height;  
    this.width = width;  
}
```

? תארו את ערכן של כל תכונות המלבן שיוצרת הפקודה הבאה:

```
Rectangle myRectangle = new Rectangle (3.2, 4.1);
```

שימו לב, כי גם ללא הוספת הקריאה *super()*, ג'אווה תוסיף באופן מובלע קריאה זו, ולכן תבצע קריאה לשיטה-הבונה חסרת הפרמטרים של המחלקה "צורה". אף-על-פי-כן, אנו נקפיד ביחידה זו על הוספה מפורשת של הפקודה *super()*.

מה היה מתבצע אילו לא היינו מוסיפים למחלקה "צורה" שיטה-בונה כלשהי? כפי שלמדנו, לכל מחלקה שלא הוגדרה בה שיטה-בונה, מוסיפה ג'אווה קונסטרקטור ברירת מחדל, והוא מאתחל את תכונות-העצם לפי ערכי ברירת המחדל שלהן. כלל זה תקף גם בעבור מחלקות מופשטות.

כך למשל, לפני שהגדרנו שיטה-בונה למחלקה "צורה", ג'אווה הגדירה שיטה-בונה שאינה מקבלת ערכים, אף שהמחלקה "צורה" היא מופשטת. אם לא נפתח את השיטות-הבונות של תת-המחלקות "מעגל" או "מלבן" בקריאה *super()*, הפקודה תיקרא באופן מובלע.


אם כן, למחלקות מופשטות, כמו לכל מחלקה, תהיה תמיד שיטה-בונה שכתבנו במפורש, או קונסטרקטור ברירת מחדל (אם לא כתבנו שיטה כזו). שימו לב, ממחלקה מופשטת לא ניתן ליצור עצמים – מה שמבדיל אותה ממחלקה רגילה. אף שבמחלקה כזו מוגדרת שיטה-בונה, לא נוכל להפעילה על ידי שימוש במילה השמורה *new*. השימוש בשיטות אלו יעשה בתת-המחלקות: כאשר אנו יוצרים עצמים מתת-מחלקות של מחלקה מופשטת, תתבצע (במפורש או במובלע) קריאה לשיטה-בונה של המחלקה המופשטת, והיא תקצה את המקומות המתאימים בזיכרון.

במחלקות רגילות הקפדנו לא להסתמך על קונסטרקטור ברירת המחדל שמספקת לנו ג'אווה. מאחר ופעמים רבות מחלקות מופשטות אינן מכילות תכונות (וממילא אי-אפשר ליצור מהן עצמים), לא תמיד נקפיד להוסיף למחלקות מופשטות קונסטרקטור מפורש – כלומר נסמוך על קונסטרקטור ברירת המחדל של ג'אווה. במקרים אחרים, כאשר למחלקה המופשטת יש תכונות שחשוב לנו לדאוג לאיתחולן, נקפיד להגדיר במחלקה המופשטת שיטה-בונה (אחת או יותר), שתאתחל את התכונות כנדרש.

ט. מחלקות מופשטות בשירות הפולימורפיזם

פולימורפיזם מאפשר להתייחס לעצמים ממחלקות שונות באמצעות טיפוס משותף. למשל, המחלקה המופשטת Shape מאפשרת לנו להתייחס לעצמים שונים (מעגלים, מלבנים וכו') באמצעות הטיפוס המשותף "צורה". אף שהשיטה draw() של המחלקה "צורה" היא שיטה מופשטת, נוכל להפעיל אותה על צורות שונות, תוך הסתמכות על העובדה שכל מופע של צורה שייך למעשה לתת-מחלקה לא מופשטת שמימשה את השיטה. אולם, לפני שנראה דוגמה לשימוש כזה בפועל, עלינו להבהיר נקודה חשובה.

אי-אפשר ליצור עצמים מהמחלקה המופשטת Shape. אולם, בהחלט ניתן להגדיר הפניה מטיפוס Shape. כאשר אנו רוצים ליצור עצם, עלינו לבנות אותו בעזרת שיטה-בונה של מחלקה שאינה מופשטת. לאחר שבנינו אותו נוכל לבצע המרה כלפי מעלה (upcasting), ולהציבו במשתנה מטיפוס Shape. בשורות הבאות תוכלו לראות דוגמאות להמרות כאלה:

 Shape shape1 = new Shape();

לא חוקי! המחלקה Shape היא מופשטת, ולכן לא ניתן ליצור ממנה עצמים

Rectangle rec1 = new Rectangle (3.4, 5.6);


חוקי: המחלקה Rectangle אינה מופשטת, ולכן ניתן ליצור ממנה עצמים

Shape shape1 = rec1;

חוקי: המלבן rec1 הוא סוג של Shape, ולכן ניתן להמיר אותו כלפי מעלה למשתנה מטיפוס Shape

Rectangle rec2 = (Rectangle)shape1;

חוקי: הצורה shape1 נוצרה כמלבן, ולכן ניתן להמיר אותה כלפי מטה ולהציבה במשתנה מטיפוס מלבן

 Circle cir2 = (Circle)shape1;

לא חוקי! הצורה shape1 נוצרה כמלבן, ולכן לא ניתן להמיר אותה כלפי מטה לטיפוס Circle

Shape shape1 = new Rectangle (3.4, 5.6);

חוקי: באגף ימין נוצר עצם מטיפוס Rectangle, ומיד הוא מומר כלפי מעלה לטיפוס Shape

בתוכנית הבאה נראה כיצד נוכל לנצל את היכולת להסתכל על המופעים של תת-המחלקות דרך הפניות מטיפוס "צורה":

```
/** This class shows a polymorphic use of abstract classes. */
public class CompareShapes {

    public static void main (String[] args){

        // Creates an array of Shapes
        Shape[] shapes = new Shape[4];

        shapes[0] = new Rectangle (4.5, 6.7);
        shapes[1] = new Circle (6.9);
        shapes[2] = new Circle (5.6);
        shapes[3] = new Rectangle (3.22, 5.7);

        double maxArea = 0;
        double currentArea = 0;
        int maxShapeIndex = 0;

        for (int i = 0; i < shapes.length; i++){
            currentArea = shapes[i].getArea();
            if (currentArea > maxArea){
                maxArea = currentArea;
                maxShapeIndex = i;
            }
        }
        //end of for

        System.out.println ("The largest shape is shape number " + maxShapeIndex);
        System.out.println ("This shape has an area of " + maxArea);
    }
    //end of main
}
//end of class
```

? מה יהיה פלט התוכנית?

המחלקה CompareShapes מכילה שיטת main(...). בשיטה זו אנו בונים מערך של ארבע צורות. שימו לב שבבניית המערך אנו משתמשים במילה *new*, אך איננו בונים עצם מסוג "צורה", אלא מערך של הפניות מסוג "צורה". בשורות הבאות אנו בונים שני מלבנים ושני מעגלים, ומציבים

אותם במערך. כאמור, פעולה זו אפשרית, שכן המחלקות מלבן ומעגל הן תת-מחלקות של Shape. לכן, ניתן להמיר את העצמים ממחלקות אלו כלפי מעלה לטיפוס "צורה". מכך אנו יכולים להבין את החשיבות של הגדרת המחלקה "צורה", למרות היותה מופשטת: לאחר שבנינו צורות שונות (מטיפוסי מעגלים, מלבנים וכו') נוכל להתייחס לכולן בשם המשותף "צורה". בשלב הבא, אנו מבקשים מכל אחת מהצורות שבמערך להחזיר את שטחה. בכל פעם השיטה שומרת את השטח הגדול ביותר, ובסוף היא מדפיסה את האינדקס במערך של הצורה הגדולה ביותר ואת שטחה.

שימו לב לפקודה:

```
currentArea = shapes[i].getArea();
```

לכאורה, נראה שזוהי קריאה לשיטה המופשטת (והלא ממומשת!) של העצם (מטיפוס "צורה") הנמצא במקום ה-i במערך shapes, אולם אין זה כך. מאחר שהמחלקה Shape היא מופשטת, אנו יודעים בוודאות שכל צורה נבנתה מלכתחילה על-ידי מחלקה לא מופשטת. בעבור כל ערך של i, המשתנה shapes[i] מכיל עצם מהמחלקות המוחשיות "מעגל" או "מלבן". מחלקות אלו יורשות מן המחלקה "צורה", ולכן בהכרח הגדירו מחדש את השיטה getArea() ומימשו אותה. הקריאה shapes[i].getArea() תזמן את אחת השיטות המוגדרות מחדש בתת-המחלקות.

בכל קריאה לשיטה getArea(), מתבצעת פעולה בעלת מימוש שונה, התלויה במחלקה שהצורה נוצרה ממנה: אם הצורה שבידינו היא עיגול, ערך הרדיוס יועלה בריבוע ויוכפל בקבוע π . לעומת זאת, אם הצורה היא מלבן, הרוחב יוכפל באורך.

שימוש זה ב"אותה שיטה", אך במימושים שונים לכל צורה, הוא הדוגמה לשימוש בפולימורפיזם. אנו אכן מתייחסים לכל הצורות דרך הטיפוס המשותף "צורה", ופונים אל השיטה המשותפת getArea(), אולם למעשה, כל צורה היא מטיפוס מפורט (מעגל או מלבן), והיא מבצעת את חישוב השטח באופן ייחודי. כאן מתאפשר הפולימורפיזם כתוצאה משימוש במנגנון הירושה הרגיל, שכן Circle ו-Rectangle הן תת-מחלקות של Shape.

י. מחלקות מופשטות ואיברים סטטיים

איך מתנהגים איברים סטטיים במחלקות מופשטות?

התשובה פשוטה: איברים סטטיים של מחלקות מופשטות מתנהגים בדיוק כמו איברים סטטיים של מחלקות רגילות – הם קיימים גם אם לא אותחל מופע מסוים כלשהו (הם הרי סטטיים – כלומר שייכים למחלקה, ולא למופע). במילים אחרות, אפשר לקרוא לתכונה סטטית או לשיטה סטטית של מחלקה מופשטת. לדוגמה, אם למחלקה Shape יש תכונה סטטית בשם background המתארת את הרקע שעליו יצוירו כל הצורות, אפשר לגשת אליה על ידי הקריאה:

```
Shape.background;
```

אם למחלקה Shape יש שיטה סטטית בשם resetBackground() המשנה את הרקע לפי ברירת מחדל מוגדרת כלשהי, הרי שניתן לזמן את השיטה הזו על ידי הקריאה:

```
Shape.resetBackground();
```

ההבדל היחיד בין מחלקות מופשטות לרגילות הוא שממחלקה רגילה ניתן ליצור מופע, ולפנות לאיבר הסטטי דרך המופע, ואילו ממחלקה מופשטת אי-אפשר לפנות לאיבר הסטטי דרך המופע. ניזכר בדוגמה שראינו בפרק 5 – "פענוח צפונות ה-main(...)":

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

```
int j = ic.counter;
```

כזכור, שם אמרנו שנעדיף את צורת הקריאה הראשונה, דרך המחלקה, ולא דרך מופע, כדי להדגיש שהתכונה או השיטה שייכות למחלקה, ולא למופע.

? אם המחלקה InstanceCounter היא מחלקה מופשטת, קריאה לאיבר סטטי כמו בדוגמה

האחרונה אינה אפשרית, מדוע?

יא. סיכום

בפרק זה נתקלנו בבעיה מעניינת: בתהליך מידול מסוים גילינו כי אנו זקוקים לעצמים מן הטיפוס "צורה". טיפוס זה מכיל תכונות ושיטות, שאת חלקן יכולנו לממש כרגיל. אף-על-פי-כן לא יכולנו להגדיר את הטיפוס "צורה" על ידי מחלקה רגילה, שכן לא יכולנו לממש את כל השיטות של הטיפוס. לו היינו מגדירים את המחלקה "צורה" כרגיל, ובתוך כך משאירים חלק מן השיטות ללא מימוש, היינו נתקלים בבעיה רצינית: המשתמש היה עלול ליצור בתום לב עצמים שאינם מתפקדים כהלכה, שכן חלק משיטותיהם אינן ממומשות.

בעזרת הגדרה של מחלקה מופשטת, יכולנו להימנע ממימוש של חלק מהשיטות. מנגנון המחלקות המופשטות אינו מאפשר למשתמש ליצור עצמים ישירות ממחלקה שאינה ממומשת עד תומה. בנוסף, מנגנון ההפשטה מבטיח לנו שמחלקות יורשות של המחלקה המופשטת, שאינן מופשטות, ימלאו את חלקן בחוזה, ויממשו את השיטות המופשטות (או יהיו מופשטות בעצמן). לפיכך, מנגנון זה מאפשר לנו ליצור מידול מתאים של הבעיה גם כאשר המושג שאנו ממדלים מופשט מכדי שנוכל לממש אותו במלואו. בנוסף, מידול זה מאפשר לנו להרחיב את השימוש בירושה ובפולימורפיזם גם לטיפוסים מופשטים, שלא ניתן להגדירם בעזרת מחלקה רגילה.

מושגים

<i>abstract</i>	מופשט
abstract method	שיטה מופשטת
abstract class	מחלקה מופשטת

פרק 7 דף עבודה מס' 1

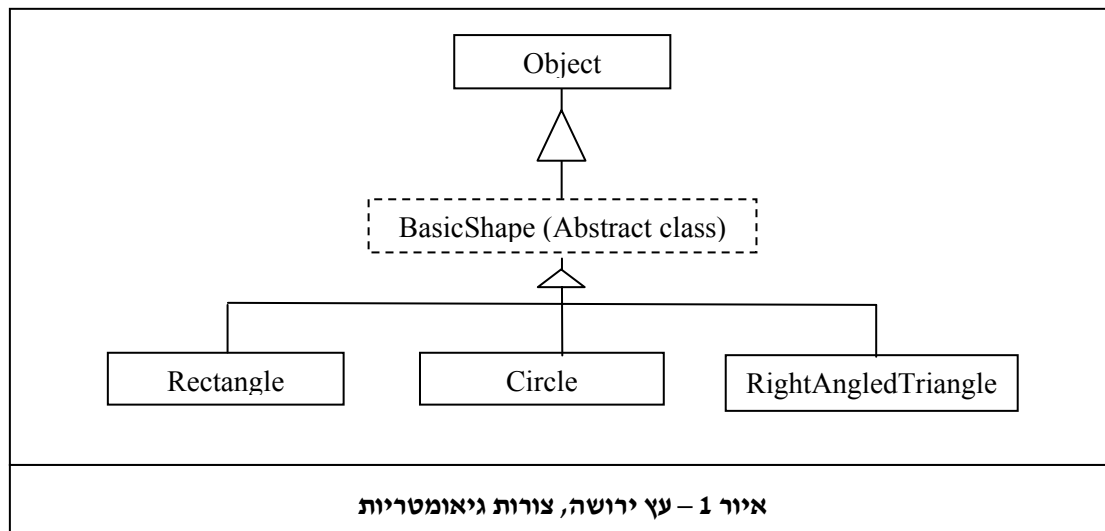
צורות גיאומטריות

מטרות

תרגול השימוש בעקרון הפולימורפיזם ויצירת מחלקות מופשטות.

מה עליכם לעשות?

עליכם לכתוב מחלקת-על מופשטת בשם "צורה בסיסית" (BasicShape) ולממש את תת-המחלקות "מלבן" (Rectangle), "מעגל" (Circle) ו"משולש ישר זווית" (RightAngledTriangle). יחסי המחלקות מתוארים בעץ הירושה הבא:



בשלב הבא עליכם לכתוב מחלקה בשם ShapeSystem שתכלול את השיטה main(...), ותאפשר למשתמש ליצור צורות גיאומטריות מסוגים שונים (ראו פירוט בהמשך). מותר לכם לשנות את התרשים המופיע באיור 1, ובלבד שיחסי הירושה המתוארים בו יישמרו.

רקע

- כדי לייצג נקודה במישור עליכם להשתמש במחלקה Point שהגדרתם בפרק 4, בדף עבודה מס' 1 ("נקודת התחלה").
- כדי לממש חלק מהשיטות תזדקקו לאיברים מתוך המחלקה Math הנמצאת ב-Java API.
- כדי לממש חלק מהשיטות ייתכן ותזדקקו לנוסחאות המופיעות בסוף דף העבודה.
- זכרו לבדוק את תקינותה של כל מחלקה שאתם מגדירים, תוך כדי העבודה עליה ובסיומה (אם הדבר אפשרי).

המחלקה BasicShape

מחלקה המייצגת צורה מופשטת. עליכם לממש את הממשק הבא :

תיאור השיטה	חתימת השיטה
שיטה-בונה היוצרת צורה לפי צבע נתון	BasicShape (String color)
מחזירה את צבע הצורה	String getColor()
מחזירה מחרוזת המבטאת את מאפייני הצורה	<i>abstract</i> String toString()
מחזירה את שטח הצורה	<i>abstract double</i> getArea()
מחזירה את היקף הצורה	<i>abstract double</i> getPerimeter()

המחלקה Rectangle

לשם מימוש תת-מחלקה של BasicShape המייצגת מלבן השתמשו במחלקה Rectangle שכתבתם בפרק 4 בדף עבודה מס' 2. שנו אותה כך שהיא תירש מ-BasicShape וכך שצבעו של מלבן שיווצר יהיה תמיד אדום (לשם כך יש לשנות את השיטה-הבונה). שנו את השיטה toString() לפי הצורך כך שתייצג את המלבן על כל מאפייניו.

המחלקה Circle

לשם מימוש תת-מחלקה של BasicShape המייצגת מעגל השתמשו במחלקה Circle שכתבתם בפרק 6 בדף עבודה מס' 4, ושנו אותה כך שהיא תירש מ-BasicShape. בנוסף, החליפו את השיטה-הבונה, כך שחתימתה תהיה :

Circle (Point center, *double* radius, String color)

שנו את השיטה toString() לפי הצורך כך שתייצג את המעגל על כל מאפייניו.

המחלקה RightAngledTriangle

תת-מחלקה זו של BasicShape מייצגת משולש ישר זווית. המשולש נבנה לפי קודקוד הזווית הישרה ואורכי הניצבים, וצבעו ירוק.

המחלקה ShapeSystem

מחלקה זו תכיל את השיטה `main(...)`, וייבנו בה צורות גיאומטריות שונות לפי קלט שיתקבל מהמשתמש. בסיום, תציג התוכנית את הפרטים של כל הצורות, כפי שמתואר בהמשך. במחלקה זו עליכם ליצור מערך, שיוכל להכיל סוגים שונים של צורות (אתחלו את גודל המערך ל-3 לפחות).

1. כדי לאפשר למשתמש ליצור צורות, הציגו חלון-פלט ובו התפריט הבא:

Select a shape:

1. Circle
2. Rectangle
3. Right Angled Triangle

2. השתמשו בחלונות הקלט כדי לקבל מהמשתמש נתונים שיאפשרו לכם ליצור את הצורה שהוא בחר.

3. צרו צורה בהתאם לקלט שהתקבל מהמשתמש, והציבו אותה במערך שהגדרתם.

4. חזרו על פעולות 2 עד 4, כל עוד המערך שהגדרתם אינו מלא.

5. הדפיסו את כל הצורות המוצבות במערך, לפי המבנה הבא:

shape <array index> is: <shape's details>

The area is: <shape's area>

The perimeter is: <shape's perimeter>

shape's details – מאפייני הצורה כפי שהוגדרו במחלקה שלה (על ידי השיטה `toString()`).

נוסחאות עזר

$S_{\text{RightAngledTriangle}} = (\text{leg1} * \text{leg2}) / 2$	שטח משולש ישר זווית
$S_{\text{rectangle}} = \text{width} * \text{height}$	שטח מלבן
$S_{\text{circle}} = \pi * R^2$	שטח מעגל
$P_{\text{circle}} = 2 * \pi * R$	היקף מעגל

איברים של המחלקה Math שמומלץ להיעזר בהם

הערה: כדי לבצע את החישובים המופיעים בטבלה תוכלו להיעזר באיברים הסטטיים הבאים המופיעים במחלקה `Math`. לפירוט מלא אודות האיברים, עיינו ב-Java API.

שם האיבר	תיאור האיבר
<code>static double PI</code>	מספר שהוא הקירוב הטוב ביותר ל- π
<code>static double pow (double a, double b)</code>	מחזירה את הערך של a בחזקת b
<code>static double sqrt (double a)</code>	מחזירה את השורש הריבועי של a

שאלות

1. האם ביצעתם המרה כדי להציב את הצורות במערך? אם כן, לאיזה טיפוס המרתם את הצורות, ואיזה סוג המרה זו?
2. האם ניתן ליצור עצמים מהמחלקה BasicShape? אם כן, הסבירו כיצד תיצרו עצם מטיפוס זה. אם לא, הסבירו מדוע.
3. מדוע הגדרנו במחלקה BasicShape שיטה-בונה?
4. מדוע לא הגדרנו את השיטה getColor() במחלקה BasicShape כמופשטת?
5. אם נרצה ליצור מחלקה שאינה אבסטרקטית שירשת מ-BasicShape, אילו איברים חובה עלינו להגדיר מחדש?
6. נניח שהיינו רוצים לשנות את הגדרתה של toString() במחלקה BasicShape, כך שתהיה שיטה רגילה (שאינה מופשטת). כיצד הייתם מממשים אותה?

בהצלחה!

פרק 7 דף עבודה מס' 2

צעצוע

מטרות

תרגול השימוש בעקרון הפולימורפיזם ויצירת מחלקות מופשטות.

מבוא:

חברת "צעצוע לי" מעוניינת למחשב את מאגר הצעצועים במחסן המרכזי של החברה. לביצוע המשימה הוגדרו שתי מחלקות בעבור שני סוגים של צעצועים: המחלקה Dolly בעבור צעצועים בצורת אדם, והמחלקה TeddyBear בעבור צעצועים בצורת דובונים.

במחלקה Dolly מוגדרים האיברים הבאים:

התכונות: שם בובה, מחיר בסיס של בובה, מספר אביזרים מצורפים, מחיר לאביזר (לכל האביזרים של אותה בובה מחיר זהה).
השיטות:

1. החזרת מחיר בובה לצרכן. מחיר בובה הוא מחיר הבסיס שלה, בתוספת מחיר האביזרים המצורפים אליה (מחושב על פי מספר האביזרים כפול המחיר לאביזר).
2. עדכון מחיר הבסיס של בובה על ידי העלאתו ב-p אחוזים.
3. יצירת מחרוזת המייצגת את הבובה בפורמט שמצוין במימוש למטה והחזרתה.

במחלקה TeddyBear מוגדרים האיברים הבאים:

התכונות: שם בובה, מחיר בסיס של בובה, משתנה בוליאני המציין אם הבובה גדולה או לא, צבע הבובה.
השיטות:

1. החזרת מחיר בובה לצרכן. מחיר בובה לצרכן נקבע על פי גודלה. המחיר לצרכן של בובה קטנה הוא מחיר הבסיס בתוספת 15%, והמחיר לצרכן של בובה גדולה הוא מחיר הבסיס בתוספת 30%.
2. עדכון מחיר הבסיס של בובה על ידי העלאתו ב-p אחוזים.
3. יצירת מחרוזת המייצגת את הבובה בפורמט שמצוין במימוש למטה והחזרתה.

```

public class Dolly{
    private String name;
    private double basePrice;
    private int accessories;
    private double accPrice;
    public Dolly (String name, double basePrice, int accessories, double accPrice){
        ...
    }
    public double computePrice(){
        ...
    }
    public void changeBasePrice (double percent){
        ...
    }
    public String toString(){
        return "Dolly " + this.name + ": the price is " + this.computePrice();
    }
}

```

```

public class TeddyBear{
    private String name;
    private double basePrice;
    private boolean isBig;
    private String color;
    public TeddyBear (double basePrice, String color, boolean isBig){
        ...
    }
    public double computePrice(){
        ...
    }
    public void changeBasePrice (double percent){
        ...
    }
    public String toString(){
        return this.color + "TeddyBear " + this.name + ": the price is "
            + this.computePrice();
    }
}

```

- א. הגדירו מחלקה חדשה – Toy, שהמחלקות Dolly ו-TeddyBear יורשות ממנה.
כתבו מחדש את הגדרת המחלקות Dolly ו-TeddyBear בהתאמה.
- ב. ממשו את שלוש המחלקות.
- ג. כתבו תוכנית ראשית המבצעת את המשימות הבאות:
1. בניית מערך ובו חמש בובות מסוגים שונים.
 2. הדפסת פרטיהן של כל הבובות ומחירן של כל חמש הבובות.

בהצלחה!

