המחלקה Object

בשפת Java כל המחלקות יורשות מהמחלקה Object.

אם בהגדרת מחלקה אין שימוש במילה השמורה extends, אזי ברירת המחדל היא שמחלקה זאת יורשת מהמחלקה Object.

לכן שתי הגדרות המחלקות הבאות שקולות:

```
class Point
{
        //whatever
}
```

ו-

```
class Point extends Object
{
        //whatever
}
```

לצורך ההדגמות להלן, נגדיר את המחלקה Point (רק את הדברים הרלוונטיים לכאן):

```
public class Point
{
        private double _x, _y;
        public String toString()
        {
                return  "(" + _x + ":" + _y + ")" ;
        }

        public boolean equals (Point other)
        {
                double dx = _x - ((Point)other)._x;
                double dy = _y - ((Point)other)._y;
                return (Math.sqrt(Math.pow(dx,2.0) + Math.pow(dy,2.0))
                        < Point.EPSILON);

        }
}
```

כיוון שכל המחלקות יורשות מהמחלקה Object, אזי כל השיטות הציבוריות של המחלקה Object יכולות להיקרא על-ידי כל אובייקט בשפה.

המחלקה Object מוגדרת בחבילה java.lang.

בתרשים הבא יש רשימה של מספר שיטות של המחלקה.

boolean equals (Object o)
השיטה מחזירה true אם האובייקט הנוכחי הוא alias של האובייקט o.

String toString ()
השיטה מחזירה ייצוג במחרוזת של האובייקט הנוכחי

תרשים 1: מספר שיטות של במחלקה Object

השיטה toString

מבלי שידענו כבר השתמשנו בשיטות של המחלקה Object בדוגמאות שראינו.

מה בעצם קרה כשהרצנו את הפקודות הבאות?

```
Point p1 = new Point(0,5);
System.out.println(p1);
```

הקומפיילר (המהדר) שמצפה לקבל בפעולת ההדפסה אובייקט מטיפוס String, מקבל אובייקט מטיפוס Point, ובאופן אוטומטי הוא קורא לשיטה toString.

לפני שמימשנו את השיטה toString במחלקה Point, הודפס פלט מהצורה הבאה:

```
Point@110c31
```

מחרוזת זאת המתארת את האובייקט p1, נוצרה במימוש של השיטה toString במחלקה Object.

במחרוזת יש שרשור של שם המחלקה שממנו נוצר האובייקט p1 הסימן שטרודל ומספר שדרכו ניתן להגיע לכתובת האובייקט בזכרון.

כאשר מימשנו את השיטה toString במחלקה Point, בעצם דרסנו (override) את השיטה של המחלקה Object. ביצענו זאת כיוון שרצינו להתאים את השיטה לצרכים שלנו וליצור מחרוזת המייצגת אובייקטים של המחלקה Point.

לאחר המימוש של השיטה toString במחלקה Point הפלט של הפקודה:

```
System.out.println(p1);
```

היה

(0.0,5.0)

השיטה equals

המטרה של השיטה לקבוע האם שני אובייקטים זהים. המימוש של השיטה במחלקה Object מחזיר true אם שני המצביעים של האובייקטים מצביעים לאותו אובייקט בזכרון (זאת אומרת האם הם aliases).

לעיתים קרובות, מחלקות דורסות את המימוש של השיטה equals במחלקה Object לטובת קביעה האם שני האובייקטים בעלי אותם מאפיינים. לדוגמא במחלקה String מומשה השיטה כך שהיא מחזירה true אם שני אובייקטים מטיפוס String מכילים את אותם תווים באותו סדר.

זאת אומרת הערך המוחזר מהפקודה בשורה שלוש שברצף הפקודות הבא הינו true:

```
String s1 = "aaa";
String s2 = " aaa";
s1.equals(s2) ;
```

<u>שים לב:</u> כמו שלמדנו ביחידה 12, בכדי לדרוס שיטה של המחלקה שממנה יורשים, חייבים שהחתימה תהיה זהה, אחרת לא תתבצע דריסה ויהיה overloading.

לדוגמא:

החתימה של השיטה equals במחלקה Point הייתה:

```
public boolean equals(Point other)
```

והיא אינה זהה לחתימה של השיטה equals במחלקה Object, שבה הפרמטר הפורמלי היה מסוג Object (ראה תרשים 1).

נתבונן בקוד הבא:

```
Point p1 = new Point(0,5);
Point p2 = new Point(0,5);
System.out.println( ( (Object)p1 ).equals( (Object)p2 ) );
```

שאלה: איזה מימוש של השיטה equals יתבצע לדעתך (של המחלקה Object או של המחלקה Point) ?

תשובה: מכיוון שעשינו casting ל- Object, והשיטה equals הממומשת במחלקה Point אינה דורסת את המימוש של המחלקה Object (ולכן לא פועל חוק הכבידה), השיטה שתתבצע היא של המחלקה Object. לכן השיטה תחזיר false כיוון ששני המצביעים אינם aliasing, הם מצביעים לאוביקטים שונים (שבמקרה בעלי אותם מאפיינים).

עכשיו, כשלמדנו את נושא הפולימורפיזם, נוכל לממש את השיטה equals במחלקה Point כך שתדרוס את המימוש של המחלקה Object.

להלן מימוש השיטה במחלקה Point:

```
public boolean equals(Object other) {
  if(other == null)
     return false;
  if(other instanceof Point)
  {
     double dx = _x - ((Point)other)._x;
     double dy = _y - ((Point)other)._y;
     return (Math.sqrt(Math.pow(dx,2.0) + Math.pow(dy,2.0))
        < Point.EPSILON);
  }
  else
  {
     System.out.println("Error: the parameter should be of type Point");
     return false;
  }
}
```

נתבונן שוב בקטע הקוד שראינו קודם:

```
Point p1 = new Point(0,5);
Point p2 = new Point(0,5);
System.out.println( ( (Object)p1 ).equals( (Object)p2 ) );
```

עכשיו, באמת תופעל השיטה equals הממומשת במחלקה Point (בזכות כח הכבידה, שגורם לביצוע השיטה הכי ספציפית של האובייקט p1), והשיטה תחזיר true, כיוון שערכי שני האובייקטים זהים.

<u>שימו לב:</u> הפקודה:

```
( (Object)p1 ).equals( (Object)p2 ) ;
```

נכתבה באופן זה רק לצרכי השאלה, הדרך הפשוטה לקריאה לשיטה equals היא בדרך שלמדנו:

```
p1.equals(p2);
```

הממשק Comparable (Interface)

לעיתים לא נסתפק בקביעה האם שני אובייקטים זהים זה לזה (כמו שמחזירה השיטה equals), ונרצה לקבוע האם אובייקט אחד הוא 'לפני' או 'אחרי' האובייקט השני.

מי שמתכנן את המחלקה צריך לקבוע מתי אובייקט אחד נמצא 'לפני' אובייקט אחר.

לדוגמא מי שכתב את המחלקה String קבע שסדר האובייקטים יקבע לפי סדר האלף בית, זאת אומרת "aaa" נמצא לפני "bbb".

בכדי לבצע משימה זאת כתבו את הממשק Comparable.

הממשק נמצא בחבילה java.lang, ומכיל רק שיטה אחת compareTo (כמובן שהשיטה מופשטת כמו כל השיטות בממשקים).

המחלקה String מממשת את הממשק Comparable וממשת את השיטה compareTo.

נסביר את הערך המוחזר מהשיטה compareTo הממומשת במחלקה String על-ידי הדוגמא הבאה:

```
String s1 = "aaa";
String s2 = "bbb";
if (s1.compareTo(s2) < 0)
  System.out.println("'aaa' is less than 'bbb'");
```

הערך המוחזר מהשיטה compareTo:

מספר שלילי - אם האובייקט s1 קטן מ-s2

אפס - אם האובייקט s1 שווה ל-s2

ומספר חיובי - אם האובייקט s1 גדול מ-s2

במקרה שלנו השיטה תחזיר מספר שלילי ולכן הפלט יהיה:

'aaa' is less than 'bbb'