שעור 1. תכנות מונחה עצמים

מחלקות ואובייקטים – חזרה למושגים בסיסיים

1. **מחלקה** היא מבנה לוגי המאגד בתוכו משתנים ושיטת תחת שם אחד.
אובייקט הוא משתנה מטיפוס של המחלקה. מכל מחלקה אפשר ליצור אובייקטים רבים.

2. **אופרטור . (נקודה)**
אופרטור **.** מהווה פעולת פנייה לאיבר (משתנה עצם או שיטה).
לדוגמה: נגדיר מחלקת Point, נגדיר משתנה מסוג Point p אזי p.x פונה ל-x השייך לאובייקט p, p.toString() מפעיל את השיטה toString() של אובייקט p.

3. **הערך null**
אם נותנים למשתנה את הערך null אז המשמעות היא שאותו משתנה לא יצביע על אף אובייקט.

4. **פונקצית toString()** מחזירה String המייצג את האובייקט. כאשר קוראים לפונקציה System.out.print() עם אובייקט, נקראת השיטה toString() באופן אוטומטי. אם לא מימשנו את השיטה יודפס על המסך כתובת האובייקט בזיכרון.

5. **מילה שמורה this**
בתוך כל שיטה ניתן לעשות שימוש במילה השמורה this. בכל רגע נתון בזמן ההרצה, this מכילה מצביע לאובייקט הנוכחי, האובייקט שממנו הופעלה אותה מתודה. כלומר, מצביע לאובייקט, שעליו המתודה פועלת. this is the final variable in java לכן המילה זו לא יכולה להופיע בצד שמאל של משוואה: this=new Object();
רוב הזמן אין צורך להשתמש במילה השמורה this. מקרים שבהם מתגלה הצורך ב- this כוללים בעיקר את שני המצבים הבאים:
א) כאשר מתודה מקבלת ארגומנט אל תוך פרמטר ששמו זהה לשם של משתנה. הדרך להבדיל בין השניים היא להוסיף את המילה this לפני השם של המשתנה.
ב) כאשר תוך כדי פעולתה של מתודה נתונה רוצים להפעיל מתודה אחרת (רוצים שאחת הפעולות שהיא תבצע תהיה הפעלתה של מתודה אחרת) ולשלוח אליה את המצביע של האובייקט שעליו המתודה הראשונה פועלת. במקרה כזה נוכל לשלוח אל המתודה השנייה את המילה this.

6. **הפשטת נתונים (Data Abstraction)** - התעלמות מפרטי המימוש של האובייקט והתרכזות במאופיינים הרלוונטיים שלו.

7. **כמוס (ריכוזיות, Encapsulation) והסתרת מידע (Information Hiding)**
המשתנים והמטודות הקשורות בתפקוד האובייקט מרוכזות במקום אחד, ופרטי המימוש חבויים מהמשתמש. כאשר אנו מפעילים מתודה של מחלקה (למשל, מתודה toString()), אנו משתמשים באופן בלתי נראה במשתנה עצם של המחלקה. לתכונה זו של ריכוז המשתנים והשיטות תחת שם אחד ושימוש במשתנים בצורה בלתי נראית קוראים encapsulation. זו התכונה הראשונה של שפה מונחת עצמים. באמצעות שתי תכונות אלו מסופקת למעשה הפשטת הנתונים (Data Abstraction) שלעיל.

8. **מודולאריות –** הפרדת ליחידות נושאיות (מודולים), כך שכל נושא מיוצג ע"י עצם.

9. **פונקציות בונות - constructors**
ה- constructor הוא שיטה שתפקידה לאתחל אובייקט חדש. שמו של constructor זהה לשם של המחלקה שבה הוא מוגדר. אסור שהשיטה תכלול את הפקודה return. אפילו לא return void. יש לשים לב לכך שאם לא מגדירים במחלקה אף constructor אז קיים בנאי המחדל (Default Constructor), אשר קיים באופן אוטומטי בכל מחלקה שמגדירים ב-JAVA כל עוד לא

הגדרנו בה constructor כלשהו. ה-Default Constructor הוא הפונקציה הבונה אשר מופעלת כאשר יוצרים אובייקט חדש מהמחלקה מבלי לשלוח ערכים בעת יצירתו, ובפעולתה מכניסה לכל אחד ממשתני המחלקה את ערך ברירת המחדל על פי טיפוס הערך שלו.

10. בנאי מעתיק (copy constructor) מיצר עותק מדויק של אובייקט - העתקה עמוקה.
אופרטור new מחזיר את מצביע לאובייקט ולא אובייקט עצמו.

11. מתודות חופפות - Method Overloading

בתוך אותה מחלקה ניתן לכתוב מספר methods בשם זהה, שמה שמבדיל ביניהן הוא מספר הפרמטרים ו/או סוגם. כדי שניתן יהיה לכתוב מספר מתודות בשם זהה חייב להיות שוני או במספר הפרמטרים או בטיפוס הערך שלהם. מתודה ששמה זהה לשמה של מתודה אחרת שכבר קיימת במחלקה נקראת בשם: overloaded method.

ניסיון לכתוב שתי מתודות זהות, למעט טיפוס הערך המוחזר שלהן לא מספיק, וגם איננו אפשרי.

12. משתנים סטטיים (משתני מחלקה) static variables (class variables)
משתנים סטטיים שמוגדרים בתוך מחלקה נקראים גם משתני מחלקה (class variables).
אלה הם משתנים שנוצרים פעם אחת בלבד (הם לא נוצרים שוב ושוב בכל אובייקט חדש), והם באים לתאר את המחלקה כולה או משהו שמשותף לכל אחד מהאובייקטים. כדי שמשתנה שמוגדר במחלקה ייחשב למשתנה סטטי יש צורך להוסיף לתחילת השורה, שבה הוא מוגדר, את המילה static.
את המשתנה הסטטי כל האובייקטים יכולים לראות. הוא גלוי לכולם. כל האובייקטים – בפנייתם למשתנה הסטטי פונים, למעשה, לאותו שטח בזיכרון. המשתנה הסטטי לא נמצא בתוך כל אחד מהאובייקטים שנוצרים. שטח הזיכרון שמשמש את המשתנה הסטטי מוקצה באופן אוטומטי (גם אם לא נוצר אף אובייקט מהמחלקה) מייד עם הרצתה של התכנית, ומייד אחר כך הוא גם מאותחל בערך ברירת המחדל שבהתאם לטיפוסו.

הגישה למשתנה הסטטי מחוץ למחלקה (כלומר: כאשר לא נמצאים בתוך מתודה ששייכת למחלקה) אפשרית מכל אחד מהאובייקטים שנוצרו מהמחלקה, וגם מהמחלקה עצמה. כדי לפנות אל המשתנה הסטטי יש לרשום את שם המשתנה (שמכיל מצביע לאובייקט מאותה מחלקה) או את שם המחלקה ואחריהם את שם המשתנה הסטטי כשמה שמפריד ביניהם היא אופרטור נקודה.

<u>אתחול משתנים סטטיים - Class Variable Initialization</u>

משתנה מסוג class variable (תזכורת: class variable הוא שם נוסף למשתנה סטטי) נוצר ומאותחל מייד עם הרצת התכנית (מייד עם העלאת הגדרתה של המחלקה לזיכרון). המשתנה הסטטי שנוצר מאותחל בערך ברירת המחדל של טיפוס הערך שלו, אלא אם צוין אחרת.

את המשתנים הסטטיים ניתן גם לאתחל במרוכז באמצעות static initializer block. יש לרשום static, לפתוח סוגריים מסולסלות, ובתוכן לרשום משפטי השמה של ערכים אל תוך משתנים סטטיים. במחלקה ניתן לכתוב יותר מ- static block אחד. יש לשים לב לכך ש- static blocks מתבצעים מייד בתחילת התכנית , ולפני שורות הקוד האחרות. כמו כן, ניתן להכניס ערך למשתנה סטטי יותר מפעם אחת, וגם ב static blocks שונים. במקרה כזה, ה-static blocks יתבצעו לפני יתר התכנית, והם יתבצעו על פי סדר הופעתם.

```
Public class A{
        private int x,y;
        private byte color;
        public static int numOfObjects;
        static{
                numOfObjects =1000;
        }
        public A(){
                numOfObjects ++;
        }
        .......................
}
```

13. מתודה סטטית - Static(Class) Method

בדומה לקיומם של משתנים סטטיים, כך גם קיימות מתודות סטטיות, אשר קיימות וניתנות להפעלה עוד לפני שבכלל נוצר איזשהו אובייקט (בדומה למשתנה סטטי, שגם הוא ניתן לשימוש עוד לפני שבכלל נוצר איזשהו אובייקט).

14. Modifiers הרשאות : private ו – public.

א. **private** - מתודה, שהרשאת הגישה שלה private, ניתן יהיה להפעיל אך ורק בתוך שורות ששייכות למחלקה.

ב. **public** - מתודות שהרשאת הגישה שלהן public ניתן להפעיל בתוך כל מחלקה.

ג. **protected** - (נראה בהמשך בפרק שדן בהורשה)

ד. **שום דבר – package friendly**

אם לא רושמים שום דבר לפני הגדרת המתודה אז המתודה תיחשב ל- package-friendly, משמע: הגישה אליו מותר רק ממחלקות ששייכות ל-package שאליו שייכת המחלקה של המתודה. המחלקות שיוכלו להשתמש בה הן רק המחלקות ששייכות ל package שלה. אם בקובץ הגדרת המחלקה לא נרשם משפט ה-package, אז המשמעות היא שהמחלקה שייכת ל Default Package. במקרה כזה, כל המחלקות ששייכות ל-Default Package יוכלו להשתמש בה.

אפשרויות הרשאות הגישה שניתן לתת למשתנים של מחלקה הן כל אחד מארבעת האפשרויות שאנו מכירים כבר, והמשמעות של כל אחת מהן זהה למשמעות שיש לה כאשר היא מוספית לפני מתודה.
ככלל, יש להשתדל להעניק את הרשאת הגישה private לכמה שיותר משתנים ששייכים למחלקה. במתן הרשאת הגישה private למשתנים שמכילים ערכים רגישים של המחלקה אפשר יהיה להבטיח שרק מתודות של המחלקה יוכלו לשנות את ערכם. כמו כן, כתיבת מתודות מיוחדות שיאפשרו גישה למשתנים אלה תיאלץ מתכנתים לגשת אליהם אך ורק דרך אותן מתודות, והיא לא תאפשר את הגישה הישירה אליהם. בדרך זו, ניתן יהיה לוודא שאותם משתנים רגישים מכילים כל העת ערכים חוקיים בלבד (חוקיים מבחינת תכנון המחלקה).
לעתים נגדיר מתודה כ- private . אנו נעשה זאת כאשר המתודה מבצעת פעילות או נותנת שירות חשוב ורגיש למחלקה עד כי לא נרצה לאפשר לכל מתכנת להפעיל אותה.

15. יצירת קבועים – final variables

הדרך ליצירת קבועים בתכנית שכתובה ב-JAVA היא באמצעות המילה השמורה final. יש לרשום final בתחילת שורת ההצהרה של המשתנה שרוצים שייחשב לקבוע, ולאתחל אותו.
ניתן להוסיף את המילה השמורה static בתחילת שורת ההצהרה על הקבוע, ובכך לקבל קבוע שאיננו שייך לאף אובייקט של המחלקה, ויחד עם זאת, כל אחד מהאובייקטים שנוצרו מהמחלקה יכולים לגשת אליו. הוספת static היא הגישה המקובלת ביצירת קבוע.

אוסף (Container) מה היתרונות של אוסף לעומת מערך?

יכולות של אוסף (Container):

- יצירה: אתחול.
- הוספה, בדיקת תקינות, גדילה אוטומטית.
- מספר איברים, איבר במקום.
- שוויון לוגי.
- שיכפול.
- אוסף יכול להכיל אובייקטים מטיפוסים שונים, למשל אוסף של צורות שיכול להכיל מעגל, ומשולש.

הצגת המחלקה <u>Object</u>:

- ❖ מחלקה ממנה יורשת כל מחלקה אחרת ב java
- ❖ מהווה בעצם מכנה משותף בין כל המחלקות.
- ❖ **Object – אבי (הטיפוסים) המורכבים:**
  כל הטיפוסים הלא בסיסיים יורשים מ- Object
- ❖ ניתן ליצור מערך Object[] arr = new Object[5]
- ❖ אפשר להכניס כל טיפוס לא בסיסי (נקודות, מחרוזות ..)
- ❖ ל Object מספר שיטות כגון toString()
- ❖ נשתמש במחלקה Object כדי להגדיר אוסף כללי: (vector)
- ❖ שיטות: הוספה, חיפוש, גודל.

<u>בעיות</u>: זיהוי טיפוסים, קומפילציה \ ריצה

```
/** this class represent a set (or a Vector) of general Objects.<br>
 *  It has the following properties: add, remove, duplicate,
elemetAt.. */
public class MyVector {
      // *** data members ***
// the first (init) size of the set.
      public static final int INIT_SIZE=10;
// the re-scale factor of this set.
      public static final int RESCALE=10;
      private static int numVectors = 0;
      private int _sp=0;
      private Object[] _objs;
      /*** Constructors: creates an empty set ***/
      public MyVector(){
            _sp=0;
            _objs = new Object[INIT_SIZE];
            numVectors++;
      }
/** returns the actual amount of Elements comtains in this set */
      public int size() {return _sp;}
/** add a Student to this set */
      public void add (Object p){
            if (p != null){
                  if(_sp==_objs.length) rescale(RESCALE);
                  _objs[_sp] = p;  // reference copy semantic.
                  _sp+=1;
            }
      }
/** returns a reference to the Student at the index, (not a copy) */
      public Object at(int p){
            if (p>=0 && p<size()) return _objs[p];
            return null;
      }
```

```java
/** this method returns a list of all the Students in this set! */
	public String toString() {
		String ans = "MyVector: |"+size()+"|\n";
		for(int i=0; i<this.size();i=i+1)
			ans += (i+1)+") "+at(i)+"\n";
		return ans;

	}
	public static int getNumVectors(){
		return numVectors;
	}
	/************** private methodes *************/
	private void rescale(int t) {
		Object[] tmp = new Object[_sp+t];
		for(int i=0;i<_sp;i++) tmp[i] = _objs[i];
		_objs = tmp;
	}

	public static void main(String[] a) {
		String s1 = "first";
		Integer i1=25;
		MyVector vec = new MyVector();
		vec.add(s1);
		vec.add(i1);
		MyVector vec2 = new MyVector();
		System.out.println(vec);
		System.out.println("number of  vectors:
"+MyVector.getNumVectors());
	}
}
```

תכנון נכון של מחלקה Class Design

להשתדל לקבוע את הרשאת הגישה private לכל משתנה

במתן הרשאת הגישה private למשתנים רבים ככל האפשר מבטיחים יותר את שלמות ונכונות נתוני האובייקט.

להשתדל לאתחל משתנים של המחלקה

למרות שמשתנים של מחלקה מאותחלים באופן אוטומטי בערכי ברירת המחדל על פי טיפוס הערך של כל אחד ואחד מהם, אתחול מכוון מקנה בהירות רבה יותר לקוד.

להשתדל להמעיט במספרם של משתנים מטיפוס בסיסי בהגדרת המחלקה

מחלקה שיש בה מספר רב מאד של משתנים מטיפוס בסיסי מעלה את השאלה שמא ניתן היה להגדיר במקום מחלקה אחת מספר מחלקות, ובכך להקנות לתכנית רמה גבוהה יותר של ודולאריות.

לקבוע מתודות שמאפשרות גישה למשתנים הפרטיים של המחלקה רק כאשר יש צורך בכך

משתנים שמופיעים בתוך הגדרתה של המחלקה עם הרשאת הגישה private – כדי לאפשר את הגישה אליהם תוך כדי פעולתן של מתודות ממחלקות אחרות יש לכתוב לפחות שתי מתודות מתאימות. האחת לצורך קבלת ערכו של המשתנה, והשנייה לצורך קביעת ערכו של המשתנה. כתיבת שתי מתודות אלה לכל משתנה שיש לו את הרשאת הגישה private גם כשאין בכך צורך פוגעת בבהירות של הקוד, ופוגעת במטרה שיש להרשאת הגישה private.

השתמש בסכימה קבועה ועקבית של הגדרת מחלקה

הקפדה על תבנית ברורה בהגדרתה של מחלקה תורמת לבהירות הקוד. כך למשל, אם מחליטים כי המשתנים של המחלקה יופיעו לפני המתודות שלה, ומקפידים על כך בכל המחלקות שמגדירים בהירות הקוד גדלה. באופן דומה, הקפדה על סגנון נאה, והערות עקביות ומסודרות באמצעות ה-javadoc – כל זה גם תורם לבהירות הקוד.

לפצל מחלקות למחלקות קטנות יותר כאשר זאת אפשרי

בדומה למה שכבר נאמר קודם, מספר רב יותר של מחלקות מקנה לתכנית רמה גבוהה יותר של מודולאריות. חשוב להשתדל לעבוד בכיוון זה.

לתת למחלקות ולמתודות שמוגדרות בתוכן שמות שמעידים על פעולתן

מתן שמות שניתן להבין מהם את תפקיד המחלקה, תפקיד המתודה וגם תפקידו של המשתנה חשוב ביותר. בהירות הקוד עוזרת גם למי שמתכנת וגם למי שרק מסתכל של התכנית הנכתבת להבין טוב יותר את פעולתה של התכנית.