מחלקות ועצמים

בקורסים הקודמים למדתם **תיכנות**. בקורס הזה נתחיל לראשונה ללמוד **הנדסת תוכנה**.

מה ההבדל? - בקורס תיכנות למדתם לפתור בעיות ספציפיות. כתבתם פונקציות באורך של כמה עשרות או מאות שורות - לא יותר. אבל במציאות. מערכות תוכנה מורכבות ממאות מיליוני שורות. כדי לנהל מערכת כל-כך מורכבת, לא מספיק לדעת לתכנת - צריך גם לדעת איך לתכנן, לתחזק ולנהל את המערכת.

בקורס הזה נלמד כמה עקרונות שיעזרו לנו להתמודד עם מערכות תוכנה מורכבות.

1. פירוק מערכת גדולה לרכיבים קטנים.
2. הפרדה בין רכיבים שונים – כל רכיב עומד בפני עצמו.
3. תיעוד כל רכיב בנפרד וכולם יחד.
4. בדיקת כל רכיב בנפרד וכולם יחד.
5. תיכנות "מתגונן" -- להתכונן מראש לשגיאות של מפתחים אחרים. "לפני עוור לא תתן מכשול".
6. שימוש ברכיבים קיימים ובדוקים – קוד פתוח. לימוד תמידי. "ישמע חכם ויוסף לקח".

תיכנות מונחה עצמים (`object-oriented programming`) הוא שיטה לפיתוח תוכנה, שבה החלוקה לרכיבים מנסה לשקף את העצמים בעולם. זאת לעומת **תיכנות מונחה תהליכים** (`procedural programming`), שבו החלוקה לרכיבים מנסה לשקף את הפעולות.

לדוגמה, נניח שאנחנו מתכננים מערכת לניהול ספריה.

- בתיכנות מונחה עצמים, העצמים יהיו כנראה: ספר, כתב-עת, סדרת ספרים, מדף-ספרים, כרטיס קורא, וכו'.
- בתיכנות מונחה תהליכים, התהליכים יהיו כנראה: הוסף ספר, השאל ספר, החזר ספר, פתח כרטיס קורא, וכו'.

עקרונות תיכנות מונחה עצמים

ישנם כמה עקרונות המאפיינים תיכנות מונחה עצמים.

- כימוס (encapsulation): שילוב בין מצב להתנהגות. בתיכנות מונחה-עצמים, לכל עצם יש גם שדות (משתנים) המתארים את המצב שלו, וגם פעולות (מתודות) המאפיינות את ההתנהגות שלו. זאת בניגוד לתיכנות מונחה תהליכים, שבה לעצמים יש רק שדות, בעוד שהמתודות פועלות על העצמים מבחוץ. החלוקה למחלקות מאפשרת לנו להפריד את התוכנה לרכיבים העומדים בפני עצמם. לכל אחד מהרכיבים הללו ניתן לכתוב תיעוד ובדיקות צמודים. כך, כשאנחנו מעבירים את הרכיבים למתכנתים אחרים, הם יכולים מייד להבין מה יש בהם ומה הם אמורים לעשות (דרך התיעוד), וגם לבדוק שהם אכן עושים את זה (בעזרת הבדיקות).
- הסתרת מידע (information hiding): הסתרת פרטי-המימוש של העצם מהעולם החיצוני כך שיהיה אפשר לשנות את המימוש בלי להפריע ללקוחות. בשפת ג'אבה, הדבר נעשה על-ידי הגדרת שדות כ"פרטיים" (private). גם מתודות פנימיות המבצעות פעולות-עזר מוגדרות כפרטיות. רק מספר קטן של מתודות, המאפיינות את השירות שהעצם נותן לעולם החיצון, מוגדרות כציבוריות (public).

- ריבוי צורה (polymorphism): מתודה עם שם אחד יכולה להתבצע בכמה דרכים שונות. הדבר מאפשר לכתוב אלגוריתם כללי, המשתמש במתודה מסויימת של עצם, ולבצע אותו על כל אחד מהעצמים בנפרד. בשפת ג'אבה, ריבוי-צורה מתבצע ע"י מנגנון הממשקים (interface) והירושה (extends) - נלמד עליהם בהמשך.

עצמים לעומת מחלקות

בחלק מהשפות לתיכנות מונחה-עצמים, ובפרט בשפת ג'אבה, מבחינים בין עצם לבין **מחלקה (class)**. מחלקה היא כמו מפרט לייצור עצמים. כל עצם חייב להשתייך למחלקה מסויימת. לכל העצמים מהמחלקה יש אותה **התנהגות**, אבל יש להם **מצב** שונה.

אגב, לא בכל השפות זה כך, למשל, בשפת ג'אבה-סקריפט (javascript), שנלמד בהמשך הקורס, ניתן להגדיר עצמים עם שדות ומתודות בלי להגדיר להם מחלקה.

שלבי עבודה

לפני שמתחילים **תיכנות** מונחה עצמים (object oriented **programming**), יש שלב של **תיכנון** מונחה עצמים (object oriented **design**). התיכנון לא בהכרח תלוי בשפת-תיכנות מסויימת - הוא בדרך-כלל נעשה בעברית, במסמכי וורד או על הלוח. שלבי התיכנון המקובלים (לא דווקא בסדר זה) הם:

1. תרשימי תיכנון, כגון תרשים מחלקות - נלמד בהרחבה בהמשך.
לכל מחלקה כותבים:
2. מחלקת בדיקה - מגדירה את תנאי ה"חוזה" שהמחלקה צריכה לקיים.
3. המחלקה עצמה.
4. בנאי (constructor) למחלקה.
5. שדות פרטיים.
6. מתודת toString - לצורך בדיקות.
7. מתודות נוספות: (א) מתודות קוראות, (ב) מתודות כותבות.
8. קריאת קוד – מציאת באגים של המתכנת.
9. הגנת קוד - התגוננות משגיאות של המשתמש.
10. סידור הקוד – refactoring.

כדוגמה לשלבי העבודה, בנינו מחלקה המייצגת **פולינום**, המורכב מאוסף של **מונומים**. ראו בדוגמאות הקוד.

סיכם: אראל סגל-הלוי.