עקרונות מתקדמים בתכנות

סיכום שיעור

שיעור 2 – תכנות מונחה עצמים

הנושאים שעליהם עברנו בשיעור מאוד מרכזיים, וישמשו בסיס להמון דברים שנלמד בהמשך. לכן מאוד חשוב שנוודא שאנחנו מבינים את הנושאים, ולמה OOP יכול לעזור לנו בכתיבת תכניות.

בשביל שכולנו נוכל ללמוד את החומר בצורה שהכי מתאימה לנו, עומדת לרשותנו אפשרות נוספת לחזרה על החומר, לא באמצעות מצגת, אלא באמצעות הידיים – כלומר באמצעות כתיבת קוד.
בתיקייה תוכלו למצוא תיקייה בשם: "חזרה - Hand on".

התיקייה מכילה קובץ מדריך שמאפשר לנו ללמוד את העקרונות של השיעור דרך בניית תוכנה בסיסית, ומאפשר לנו לחזור על החומר באמצעות כתיבת קוד.
שימו לב, זה ממש **לא חובה**, וזה גם לא תרגיל, אלא פשוט דרך נוספת ללמוד את אותם נושאים בדרך קצת שונה, למי שאוהב/ת לתכנת קצת יותר מלעבור על מצגת.

אם החלטתם/ן לנסות את שיטת ה-Hands On אז עברו על הקוד, וודאו שהכל מובן ושברור למה עושים את הדברים את הדברים בצורה הזו.
יכול להיות שתמצאו דרך טובה או אלגנטית יותר לעשות את אותם דברים, נסו אותה!

בנוסף אם יש חלק תיאורטי שמרגיש לכם/ן שלא יושב טוב, תזכרו שתמיד אפשר לפנות למדריך/ה!

בתרגיל הבית תידרשו לכתוב תכנית המייצגת רשת חברתית.

נושאים שעברנו בשיעור:

- תכנות פרוצדורלי VS תכנות מונחה עצמים:
  - **הגישה הפרוצדורלית** – התכנית מורכבת ממשתנים, ופונקציות שעושות פעולות עליהם.
    - לפעמים לא צריך יותר.
    - לפעמים התכנית נהיית מסורבלת (קוד ספגטי) וצריך לחשוב אם הגישה הזו היא המתאימה.
  - **גישת OOP (Object Oriented Programmin)** – אנו ממדלים את התכנית שלנו כאוסף של ישויות שלכל אחת יש מאפיינים והתנהגות
    - כאשר הגיוני לקבץ קבוצת פונקציות ומשתנים לישויות נפרדות
      - לדוגמא student ו- class
    - כאשר ישות אחת יכולה להחזיק בכל המשתנים והפונקציות אז כנראה אפשר לעבוד בגישה הפרוצדורלית.

- מחלקות:
  - תבניות שעל פיהן נוצרים אובייקטים
    - מחלקה היא סדרה של הצהרות והגדרות – מופשטת
    - מגדירים בקובץ header
  - טיפוס נתונים מורכב
    - יכולות להחזיק טיפוסים פשוטים (int, float ומצביעים למינהם)
    - יכולות להחזיק טיפוסים מורכבים (struct-ים או אובייקטים אחרים)
    - יכולות להחזיק פונקציות
  - מחלקה צריכה להיות אחראית על משהו בודד
    - אם מחלקה עושה המון דברים שונים זה סימן שיש לנו בעיה בעיצוב (design).

- namespaces:
  - מאפשר איגוד של כמה פונקציות ומתודות תחת שם אחד
    - מחלקה מהווה namespace
  - אפשר להשתמש (לייבא) שמות מתוך namespace באמצעות הפקודה **using**
    - להימנע מלייבא יותר ממה שאנו צריכים (namespace pollution).

- אובייקטים
  - מופע של מחלקה (instance)
    - **מצב של אובייקט** – ערך השדות שלו
      - מה שמבדיל בין אובייקטים זהו המצב שלהם
    - **התנהגות** – המתודות (פונקציות) שלו
      - המימוש של המתודות זהה בין כל האובייקטים
  - הקומפיילר מוסיף בצורה מרומזת (בלי שאנחנו רואים) את המצביע **this** (מצביע לאובייקט שקרא למתודה) ומעביר באמצעות הקריאה את הכתובת של האובייקט
  - **אובייקטים תופסים זיכרון**
    - בדר"כ כסכום גודל השדות
      - לא תמיד נכון בגלל padding ו- alignment

- כימוס (Encapsulation):
  - מאפשר למחלקה לחשוף החוצה רק את המתודות/שדות שהיא רוצה.
  - מצייני גישה
    - public (ציבורי) – כולם יכולים לגשת
    - private (פרטי) – ניתן לגשת רק מתוך ה namespace של המחלקה
  - נקפיד על הקונבנציה הבאה:
    - שדות המחלקה יהיו פרטיים
    - המתודות שמרכיבות את ממשק המחלקה יהיו ציבוריות
      - פונקציות עזר פנימיות הן לא חלק מהממשק כלומר יהיו פרטיות.
  - Getters/Setters – מאפשר לחשוף החוצה את ערכי השדות בתנאים שלנו
    - אפשר לבצע בדיקת קלט ולא לאפשר הכנסת ערכים לא חוקיים
    - אפשר לחשוף החוצה העתק ולא את הכתובת האמיתית של המצביע
      - לדוגמא באופרטור החיבור + מחזירים את this* כתוצאת הביניים וזה מאפשר חיבור בשרשרת.