תכנות מכוון עצמים ו- C++
יחידה 02
מחלקות: תכונות, שיטות, הרשאות

קרן כליף

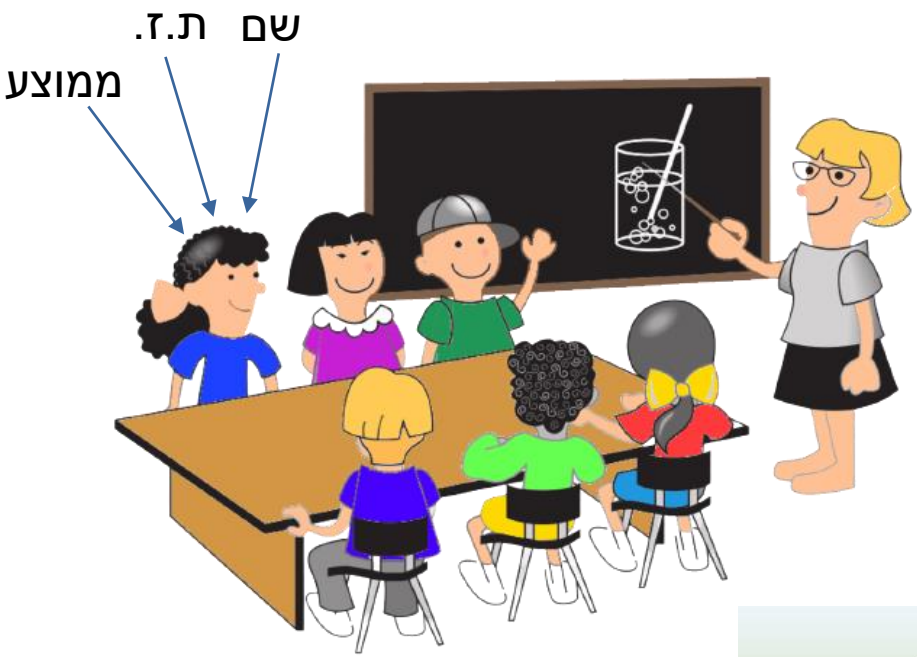
ביחידה זו נלמד:

- מהו תכנות מונחה עצמים
- מהי מחלקה
- פניה אל תכונות ושיטות המחלקה
- הרשאות גישה
- שיטות set ו- get
- ערכי default לפרמטרים של שיטות

מחלקות

תכנות מונחה עצמים

- הבסיס התפיסתי של תכנות מונחה עצמים הוא שהעולם מאורגן בעצמים (אובייקטים)
  - כל דבר הוא אובייקט:



- לכל אובייקט יש מאפיינים המייצגים אותו
- לכל אובייקט יש פעולות שהוא יודע לעשות

שאלה

- הכיסא שאתם יושבים עליו, האם הוא מחלקה או אובייקט?
- והלוח?

- **זכרו: מחלקה היא רק תאור מופשט!**

מהן תכונות ומהן שיטות ב- OOP

- **תכונה (attribute)** היא **משתנה** המשויך לאובייקט מסוים
- **התכונה** יכולה להיות מכל טיפוס שלמדנו עד כה (וטיפוסים נוספים)
- **שיטה (method)** היא פונקציה, אך משויכת לאובייקט
  - שם השיטה מעיד "מה" השיטה עושה
  - אוסף הפעולות בשיטה מעיד על ה"איך" השיטה עושה זאת
- למחלקות שונות יתכנו תכונות ושיטות עם שמות זהים
  - למשל, גם למחלקה "שחקן כדורסל" וגם למחלקה "ליצן" יכולה להיות התכונה name, ולשתיהן יכולה להיות השיטה show

OOP בשפת C++

- כאשר נכתוב מחלקה נייצר 2 קבצים:
  - קובץ עם סיומת h שיכיל את ה"מה" של המחלקה:
    - מה שמה
    - מה תכונותיה
    - מה הפעולות (שיטות, methods)
  - קובץ עם סיומת cpp שיכיל את ה"איך" של המחלקה":
    - מימוש הפעולות
- ניתן לכתוב את המימושים בקובץ ה- h, אך נעדיף להשאירו "נקי" עם תיאורי ה"מה" בלבד

דוגמה המחלקה Clock

נשים לב שזוהי רק הגדרת המחלקה, עדיין לא יצרנו משתנה!

clock.h

```
#ifndef __CLOCK_H
#define __CLOCK_H

class Clock
{
public:
    int hours, minutes;

    void show();
    void tick();
    void addMinutes(int add);
};

#endif // __CLOCK_H
```

המילה class שמורה בשפה ומעידה שפה תהייה הגדרה של מחלקה

שם המחלקה

נסביר בהמשך...

תכונות המחלקה

שיטות המחלקה

לא לשכוח ; בסוף המחלקה

clock.cpp

```
#include <iostream>
using namespace std;

#include "clock.h"

void Clock::show()
{
    cout << (hours < 10 ? "0" : "")
         << hours << ":"
         << (minutes < 10 ? "0" : "")
         << minutes;
}

void Clock::tick()
{
    addMinutes(1);
}

void Clock::addMinutes(int add)
{
    minutes += add;
    hours += minutes / 60;
    minutes %= 60;
    hours %= 24;
}
```

יש לעשות include לקובץ ה- h של המחלקה

במימוש יש לשייך את השיטה למחלקה, ע"י ציון: <class name>::

דוגמה שימוש ב- main המחלקה Clock

- כדי לפנות לתכונה או לשיטה של האובייקט נשתמש ב- "."
- לאובייקט המפעיל שיטה נקרא "אוביקט מפעיל"

```
#include <iostream>
using namespace std;

#include "clock.h"

void main()
{
    Clock c;

    c.show();
    cout << endl;

    c.hours = 22;
    c.minutes = 15;

    c.show();
    cout << endl;

    c.addMinutes(65);

    c.show();
    cout << endl;
}
```

שם הטיפוס

שם המשתנה

קריאה לשיטה של האובייקט

מתן ערכים לשדות האובייקט

קריאה לשיטה של האובייקט המקבלת פרמטרים

```
0-858993460:0-858993460
22:15
23:20
```

יצירת כמה אובייקטים מאותה מחלקה

```
#include <iostream>
using namespace std;

#include "clock.h"

void main()
{
    Clock c1, c2;

    c1.hours = 22;
    c1.minutes = 15;

    c2.hours = 14;
    c2.minutes = 45;

    cout << "Clock 1: ";
    c1.show();
    cout << endl;

    cout << "Clock 2: ";
    c2.show();
    cout << endl;
}
```

כאשר פונים לשיטה של אובייקט,
האובייקט מכיר את ערכי תכונותיו שלו בלבד

```
Clock 1: 22:15
Clock 2: 14:45
```

יצירת מערך של Clock

```
#include <iostream>
using namespace std;

#include "clock.h"

void main()
{
    Clock myClocks[2];

    myClocks[0].hours = 22;
    myClocks[0].minutes = 15;

    myClocks[1].hours = 14;
    myClocks[1].minutes = 45;

    cout << "Clock 1: ";
    myClocks[0].show();
    cout << endl;

    cout << "Clock 2: ";
    myClocks[1].show();
    cout << endl;
}
```

 hours = 22 minutes = 15	 hours = 14 minutes = 45

בדיוק כמו עבודה עם מערך של מבנים..

```
Clock 1: 22:15
Clock 2: 14:45
```

הקצאה דינאמית של אובייקט לעומת מערך Clock

```
#include <iostream>
using namespace std;

#include "clock.h"

void main()
{
    Clock* c = new Clock;

    c->hours = 22;
    c->minutes = 15;

    c->show();
    cout << endl;

    delete c;
}
```

hours = 22
minutes = 15

hours = 22 minutes = 15	hours = 14 minutes = 45

הקצאה דינאמית של אובייקט אחד

כאשר המשתנה הוא מצביע, הפניה לשדות היא באמצעות ->

שחרור ההקצאה

```
#include <iostream>
using namespace std;

#include "clock.h"

void main()
{
    Clock* c = new Clock[2];

    c[0].hours = 22;
    c[0].minutes = 30;

    c[1].hours = 20;
    c[1].minutes = 45;

    for (int i = 0; i < 2; i++)
    {
        c[i].show();
        cout << endl;
    }

    delete []c;
}
```

הקצאה דינאמית של מערך

שחרור המערך

הרשאות ושיטות set ו- get

private

public

הרשאות private ו- public

- לא נרצה שהתכונות שלנו יהיו חשופות ושכל אחד יוכל לשנות את ערכיהן ע"י השמה
  - למשל שלא יוכלו לשים בערך של הדקות מספר שאינו בין 0 ל- 59
- לכן ניתן לתת הרשאות לתכונות ולשיטות של המחלקה
  - לתכונות המחלקה ניתן הרשאת private, משמע ניתן לגשת אליהן רק מתוך המחלקה
  - שיטות יהיו תחת הרשאת public כדי שיהיו נגישות מחוץ למחלקה
  - שיטות שנרצה שיהיו לשימוש פנימי של המחלקה נגדיר ב- private

clock.h

```
#ifndef __CLOCK_H
#define __CLOCK_H

class Clock
{
private:
    int hours, minutes;

public:
    void show();
    void tick();
    void addMinutes(int add);
};
#endif // __CLOCK_H
```

הרשאת ברירת המחדל היא private, לכן לא היינו חייבים לציין אותה בהתחלה.
לכן חשוב היה בדוגמאות הקודמות לכתוב את ה- public

הרשאות private ו- public הבעיה בעקבות עדכון המחלקה

- עם שינוי ההרשאה נקבל שגיאת קומפילציה:

C2248	'Clock::hours': cannot access private member declared in class 'Clock'
C2248	'Clock::minutes': cannot access private member declared in class 'Clock'

```
#include <iostream>
using namespace std;

#include "clock.h"

void main ()
{
    Clock c;

    c.hours = 22;
    c.minutes = 15;

    c.show();
    cout << endl;
}
```

הרשאות private ו- public הפתרון בקוד המחלקה

- כדי לאפשר השמת ערך בתכונות שהן private, נכתוב שיטות שהן public המבצעות את פעולת ההשמה
  - נרצה לכתוב שיטה המקבלת כנתון את הערך המבוקש, השיטה תבצע את בדיקות התקינות על ערך זה ולבסוף תשים אותו בתכונה
  - נרצה לכתוב שיטה המחזירה את ערך התכונה

- נהוג לקרוא לשיטות אלו setter'ים ו- getter'ים

שיטות set ו- get שימוש

```
#include <iostream>
using namespace std;

#include "clock.h"

void main()
{
    Clock c;

    c.show();
    cout << endl;

    c.setHours(8);
    c.setMinutes(20);

    c.show();
    cout << endl;

    cout << "The hour is around " << c.getHours() << endl;
}
```

```
0-858993460:0-858993460
08:20
The hour is around 8
```

שיטות set ו- get מימושים מתוקנים

```
#include "clock.h"

int Clock::getMinutes() {...}
int Clock::getHours() {...}

void Clock::setMinutes(int m)
{
    if (m < 0 || m >= 60)
        cout << "Minutes have to be between 0-59. Value is unchanged.\n";
    else
        minutes = m;
}
void Clock::setHours(int h)
{
    if (h < 0 || h >= 24)
        cout << "Hours have to be between 0-23. Value is unchanged.\n";
    else
        hours = h;
}

void Clock::show() {...}
void Clock::tick() {...}
void Clock::addMinutes(int add) {...}
```

בתוך ה- setter'ים באה לידי ביטוי העבודה שהתכונות הן private: לא ניתן לשנות אותן ללא בקרה

שיטות set ו- get תוצר ההרצה לאחר התיקון

```
#include <iostream>
using namespace std;

#include "clock.h"

void main()
{
    Clock c;

    c.show();
    cout << endl;

    c.setHours(8);
    c.setMinutes(20);

    c.show();
    cout << endl;

    c.setHours(25);
    c.setMinutes(30);

    c.show();
    cout << endl;
}
```

```
0-858993460:0-858993460
08:20
Hours have to be between 0-23. Value is unchanged.
08:30
```

וכיצד ה- main ידע האם לקלוט נתונים מחדש?

- במימוש הנוכחי בחרנו להדפיס למסך כאשר הקלט שהתקבל אינו תקין
- לכן ב- main אין לנו דרך לדעת האם פעולת ה- set בוצעה בהצלחה או האם נשאר הערך הקודם
- לכן נתקן את שיטות ה- set כך שיחזירו תשובה מסוג bool, כאינדיקציה להצלחה הפעולה

שיטות set עדכון

```
#ifndef __CLOCK_H
#define __CLOCK_H

class Clock
{
private:
    int hours, minutes;

public:
    int getMinutes();
    int getHours();

    bool setMinutes(int m);
    bool setHours(int h);

    void show();
    void tick();
    void addMinutes(int add);
};

#endif // __CLOCK_H
```

```
bool Clock::setMinutes(int m)
{
    if (m < 0 || m >= 60)
    {
        cout << "Minutes … unchanged.\n";
        return false;
    }
    else
    {
        minutes = m;
        return true;
    }
}
```

```
bool Clock::setHours(int h)
{
    if (h < 0 || h >= 24)
    {
        cout << "Hours … unchanged.\n";
        return  false;
    }
    else
    {
        hours = h;
        return true;
    }
}
```

שיטות set שימוש ב- main

```
void main()
{
    Clock c;
    bool res;
    int hours, minutes;

    do {
        cout << "Enter hours: ";
        cin >> hours;
        res = c.setHours(hours);
    } while (res != true);

    do {
        cout << "Enter minutes: ";
        cin >> minutes;
        res = c.setMinutes(minutes);
    } while (res != true);

    c.show();
    cout << endl;
}
```

```
Enter hours: 30
Hours have to be between 0-23. Value is unchanged.
Enter hours: 20
Enter minutes: 100
Minutes have to be between 0-59. Value is unchanged.
Enter minutes: 30
20:30
```

שיטות set ו- get סיכום

- כדי לאפשר השמת ערך בתכונה שהיא private, נכתוב שיטת set, שהיא public, המבצעת את פעולת ההשמה:
  - השיטה תקבל כנתון את הערך המבוקש
  - השיטה תבצע את בדיקות התקינות על ערך שהתקבל
  - לבסוף השיטה תעדכן את הערך המתאים בתכונה
- כדי לאפשר קבלת ערך תכונה שהיא private, נכתוב שיטת get שהיא public
- נהוג לקרוא לשיטות אלו setter'ים ו- getter'ים

class לעומת struct

- על-מנת להגדיר טיפוס חדש:
  - בשפת C משתמשים ב- struct
  - בשפת C++ משתמשים ב- class

- ההבדל ביניהם הוא בהרשאה:
  - הרשאת ברירת המחדל ב- struct היא public ולכן יכולנו לפנות לשדותיו באופן ישיר
  - הרשאת ברירת המחדל ב- class היא private

- ניתן לכתוב גם מתודות ב- struct אך זהו כבר ממש תכנות מונחה עצמים ולכן מקובל להשתמש ב- class

האם תמיד התכונות יהיו ב- private?

- ישנם מקרים בהם אין צורך בבדיקות תקינות עבור הערכי התכונות
  - למשל, עבור המחלקה Person המייצגת אדם, שתכונותיה הם שם ו- ת.ז., כאשר אין כרגע צורך בהגבלות תקינות

```
#ifndef __PERSON_H
#define __PERSON_H

class Person
{
public:
    char name[20];
    int id;
};
#endif // __PERSON_H
```

```
#include <iostream>
using namespace std;

#include "person.h"

void main()
{
    Person p;

    strcpy(p.name, "gogo");
    p.id = 111;
}
```

האם תמיד התכונות יהיו ב- private? (2)

- אבל אם פתאום נחליט שמספר ת.ז. חייב להיות 9 ספרות, נצטרך לעדכן את המחלקה, וה- main כבר לא יתקמפל!

```
#ifndef __PERSON_H
#define __PERSON_H

class Person
{
public:
    char name[20];
private:
    int id;

public:
    int getId();
    bool setId(int i);
};
#endif // __PERSON_H
```

```
bool Person::setId(int i)
{
    int counter = 0;
    int temp = i;
    while (temp > 0)
    {
        temp /= 10;
        counter++;
    }

    if (counter == 9)
    {
        id = i;
        return true;
    }
    else
        return false;
}
```

```
void main()
{
    Person p;

    strcpy(p.name, "gogo");
    p.id = 111;
```

C2248 'Person::id': cannot access private member declared in class 'Person'

```
int Person::getId()
{
    return id;
}
```

תכונות תמיד יהיו private!

- אפילו אם בעת כתיבת המחלקה אין הגבלות תקינות על תכונה, תמיד נשים אותה עם הרשאת private ונספק מתודות set ו- get

- 2 סיבות מרכזיות לאופן כתיבה זה:

1. שאם בעתיד תתווסף בדיקת תקינות, מי שמשתמש במחלקה לא יצטרך לשנות את הקוד שלו, שכן כבר מראש הוא ישתמש במתודת ה- set, וכל שינוי בתוך גוף המתודה יהיה שקוף מבחינתו
2. אחידות, כך שהפניה לכל התכונות במחלקה תהייה באופן זהה, באמצעות שיטות set או get

אתחול תכונות שהן private

- בהמשך נראה כיצד ניתן לתת ערכים לתכונות האובייקט עם אתחולו
  - כרגע עם יצירת האובייקט כל ערכי שדותיו הם זבל
- נקפיד שתכונות האובייקט יהיו private!

ערכי default לפרמטרים

clock.h

```
#ifndef __CLOCK_H
#define __CLOCK_H

class Clock
{
private:
   int hours, minutes;

public:
    int getMinutes();
    int getHours();

    bool setMinutes(int m=0);
    bool setHours  (int h=0);

    void show();
    void tick();
    void addMinutes(int add);
};

#endif // __CLOCK_H
```

זוהי גם העמסת שיטות, שכן עכשיו ניתן לקרוא לשיטות אלו ב- 2 אופנים

```
0-858993460:0-858993460
05:00
```

```
#include <iostream>
using namespace std;

#include "clock.h"

void main()
{
    Clock c;

    c.show();
    cout << endl;

    c.setHours(5);
    c.setMinutes();

    c.show();
    cout << endl;
}
```

ביחידה זו למדנו:

- מהו תכנות מונחה עצמים
- מהי מחלקה
- פניה אל תכונות ושיטות המחלקה
- הרשאות גישה
- שיטות set ו- get
- ערכי default לפרמטרים של שיטות