

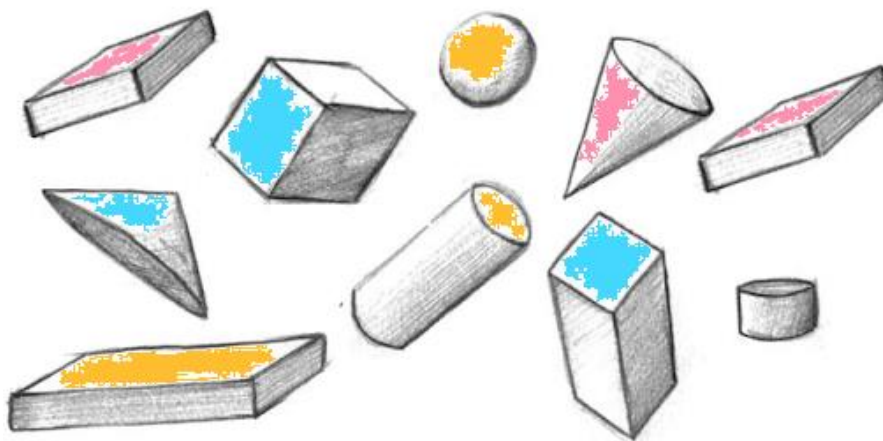
תרגול 5 צייר צורות

רקע

בשיעור למדנו על עיקרון הפולימורפיזם המאפשר לנו להתייחס לאובייקטים לפי ממשק משותף שהם חולקים. ראינו שניתן להצביע על מחלקות יורשות באמצעות מצביע מהסוג של המחלקה ממנה יורשים ולהשתמש במנגנון הקישור הדינאמי שידאג להפעיל את המימוש המעודכן של המתודה.

מטרה

בתרגיל נממש צייר גרפי, שפועל לפי פקודות טקסטואליות מהמשתמש .



אלה השלבים שנעבור:

שלב בונס	הוספת תפריט	מימוש המחלקות	כתיבת שלד
<ul style="list-style-type: none">שינוי צבעי הצורותבונס קשה במיוחד - RAIL	<ul style="list-style-type: none">שימוש בווקטור	<ul style="list-style-type: none">מימוש מחלקות התרגיל.	<ul style="list-style-type: none">הוספת שדות תיקון חתימות המתודות.

נתרגל מיומנויות חשובות:

- נתרגל ירושה בין מחלקות
- נתרגל פולימורפיזם
- נתרגל את עיקרון דריסת פונקציות
- את התרגיל צריך להגיש ב-GIT: [לינק להוראות שימוש ב-GIT](#). כדאי לקרוא גם [דגשים לתכנות נכון](#).



הידעת?

"PRACTICE MAKES PERFECT"

בהצלחה יא אלופות ואלופים!

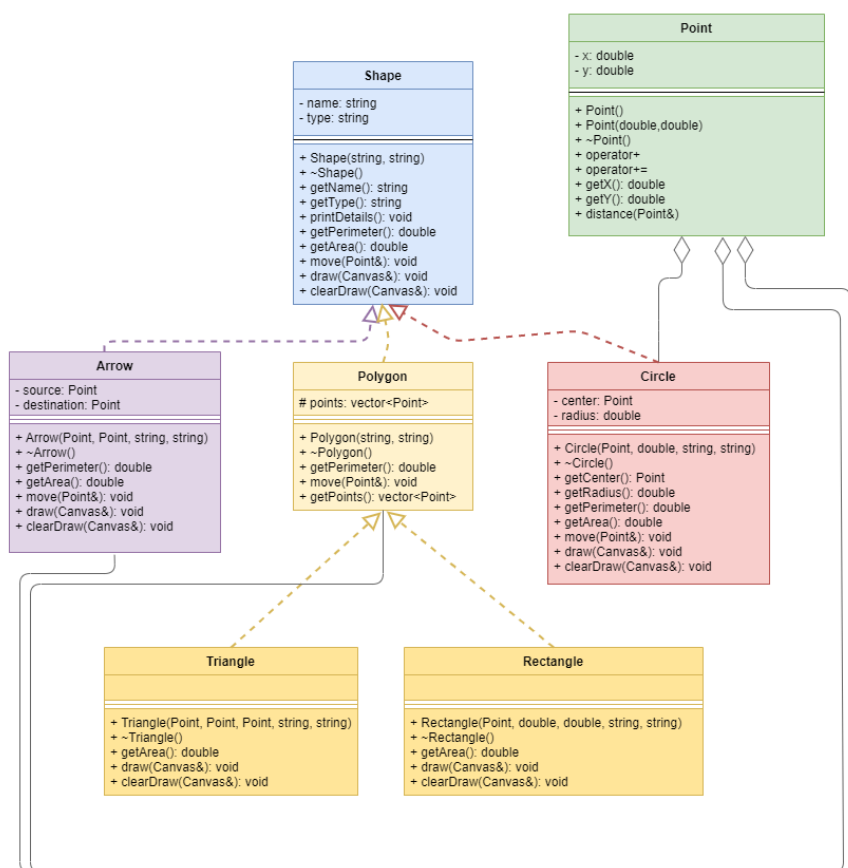
שלב 1: כתיבת שלד

שלב 4 שלב בונס	שלב 3 הוספת תפריט	שלב 2 מימוש המחלקות	שלב 1 כתיבת שלד
-------------------	----------------------	------------------------	--------------------

להלן ה-UML של מחלקות בתרגיל.



Shapes



Legend

Inheritance

Composition

שימו

המחלקה Canvas אינה מופיעה ב-UML, אולם היא חלק חשוב מהתרגיל - נשתמש בה בכל פעם

שנרצה לצייר צורה על המסך.

עברו על קבצי ה-Header שסופקו יחד עם הוראות התרגיל ובצעו את המשימות הבאות:

1. הוסיפו שדות (במידת הצורך).
2. הוסיפו `const` היכן שצריך (גם בסוף מתודה וגם על פרמטרים המועברים למתודה).
3. העבירו אובייקטים כ-reference היכן שניתן.
4. הוסיפו את המילה `virtual` בתחילת ההצהרה על מתודות שבהן נרצה שיופעל קישור דינאמי
5. הוסיפו `= 0` בסוף כל מתודה וירטואלית **טהורה**.

מתודות וירטואליות טהורות - תזכורת

- מתודה וירטואלית טהורה היא כזו שחייב לממש אותה על מנת לאפשר יצירת מופע של המחלקה.
 - לא חייב לממש מתודה וירטואלית טהורה במחלקה בה היא מוגדרת.
 - ניתן להגדיר מתודות וירטואליות טהורות כדי לחייב את המחלקות היורשות לממש פעולות מסוימות בהתאם לממשק שנקבע במחלקת האב.
6. הוסיפו את המילה `override` במתודות של המחלקות היורשות כאשר מדובר במתודה הדורסת מתודה במחלקת האב.

שימו 

אין צורך לשנות שום חתימה הקשורה למתודות `draw`, `clearDraw`.

הנחיות והערות חשובות:

- יש לכלול את הקבצים `Cimg.h`, `Canvas.h/cpp` ולהוסיפם לפרויקט ה-VS.
- המחלקה `Rectangle` מוגדרת בתוך `namespace` חדש בשם `myShapes`. הסיבה לכך היא שהשם `Rectangle` כבר תפוס במרחב השמות הגלובלי, ולכן נצטרך לממש אותו במרחב שמות חדש.
לכן:
 - כשנאלץ ליצור מלבן נשתמש בסוג `Rectangle::` `myShapes`
 - לחלופין ניתן לבצע הגדרה גלובלית בקובץ ה-Menu ולרשום:
`using myShapes::Rectangle`

שלב 2: מימוש המחלקות

שלב 4 שלב בונס	שלב 3 הוספת תפריט	שלב 2 מימוש המחלקות	שלב 1 כתיבת שלד
-------------------	----------------------	------------------------	--------------------

ממשו את המחלקות השונות:

1. מחלקת **Point**

המחלקה מייצגת נקודה במישור.

נקודה מוגדרת ע"י שתי קואורדינטות (x, y)

מתודות:

- למחלקה בנאי *default* אשר לא מקבל כלום – הבנאי מאתחל את הנקודה ל- $(0,0)$
- למחלקה יש בנאי המקבל את ערכי הקואורדינטות.
- למחלקה יש תמיכה בשני אופרטורים:
 - אופרטור $+$ - מחזיר נקודה (*copy*) בה הקואורדינטות הן סכום הקואורדינטות של הנקודות.
 - אופרטור $+=$ - מחבר את ערכי הקואורדינטות של הנקודה שהתקבלה כפרמטר אל הנקודה עליה הופעל האופרטור. מחזיר הפניה לנקודה.
- למחלקה יש מתודה *distance* אשר מחזירה את המרחק ממנה לנקודה אחרת.

2. מחלקת **Shape**

מחלקה אבסטרקטית המייצגת צורה כללית.

לכל צורה יש שם וסוג.

מתודות:

- כל צורה תומכת בפעולות הבאות:
 - הדפסת פרטי הצורה
 - חישוב היקף (*Perimeter*)
 - חישוב שטח (*Area*)
 - הזזת השורה בנקודה קיימת.
- הזזה משמעותה הוספת ערכי ה- X וה- Y של הנקודה שהתקבלה לכל נקודות הצורה.
- ציור ומחיקה

3. מחלקת **Arrow**

מחלקה המייצגת קו ישר.


הערות:

- במידה והתקבלו שתי נק' זהות יש להדפיס הודעת שגיאה ולצאת מן התכנית (אפשר להשתמש ב- `exit(1)`)
- שטח של קו ישר הוא 0.

4. מחלקת **Circle**

מחלקה המייצגת מעגל.

הערות:

- במידה והתקבל רדיוס שלילי יש להדפיס הודעת שגיאה ולצאת מן התכנית (אפשר להשתמש ב- `exit(1)`)
- אנחנו נשתמש בערך 3.14 בשביל π . בראש הקובץ תוכלו למצוא הגדרה: `#define PI 3.14`
- שימו , בבקשה השתמשו בערך 3.14 ואל תייבאו ערך PI מספריית `math/cmath`, באופן כללי זה רעיון מצוין לקבל מספר מדויק יותר (עם יותר ספרות אחרי הנקודה), אבל הטסטים שבהם המדריך/ה משתמש/ת יניחו שהערך של π הוא 3.14.

5. מחלקת **Polygon**

מחלקה המייצגת מצולע כללי.

ניתן להגדיר מצולע כאוסף של נקודות במרחב (בין כל נקודה יש צלע וכך נוצר המצולע).

הערות:

- המחלקה מחזיקה את הנקודות של המצולע במבנה נתונים מסוג `std::vector` <https://en.cppreference.com/w/cpp/container/vector>
 - כשמגדירים `std::vector` יש לציין את סוג האיברים אותו הוא מחזיק, לדוגמא במחלקה `Polygon` הווקטור שמכיל את הנקודות מוגדר כך `std::vector<Point>`
 - ניתן להוסיף איבר לסוף הווקטור באמצעות הפונקציה `push_back` המוכרת לנו מתרגיל 3.
 - את מס' האיברים בווקטור ניתן לקבל באמצעות הפונקציה `size()`.
 - ניתן לגשת לאיברי הווקטור באמצעות האופרטור `[]` (בדומה למערך).
 - ניתן למחוק איבר מתוך הווקטור באמצעות הפונקציה `vector::erase`, יש להעביר את מיקום האיבר אותו נרצה למחוק, הדרך לעשות זאת היא באמצעות העברת `iterator`. על הנושא נלמד בשיעור 10, בינתיים ניתן להשתמש במתודה בצורה הבאה:
 - בהינתן ווקטור `vec`, אם נרצה למחוק את האיבר ה- i נרשום:
`vec.erase(vec.begin() + i)`

6. מחלקת `Triangle`

מחלקה המייצגת משולש.
משולש ניתן להגדיר ע"פ 3 נקודות במרחב.

הערות:

- במידה ו-3 הנקודות שהתקבלו בבנאי המחלקה אינן יוצרות משולש (למשל כולן על אותו ישר), יש להדפיס הודעת שגיאה ולצאת מן התכנית (אפשר להשתמש ב- `exit(1)`)

7. מחלקת `Rectangle`

מחלקה המייצגת מלבן.
מלבן ניתן להגדיר ע"פ 4 נקודות במרחב.
את 4 הנקודות ניתן לחשב באמצעות נקודה אחת, אורך ורוחב.

הערות:

- לא ניתן להגדיר את שם המחלקה כ-`Rectangle` במרחב השמות הגלובלי, ולכן המחלקה מוגדרת בתוך מרחב שמות חדש בשם `myShapes`.

○ כדי ליצור מלבן חדש יש להשתמש בסוג `Rectangle::` `myShapes`

הנחיות והערות חשובות:

- בקבצים שניתנו עם הוראות התרגיל לא רשומות החתימות המדויקות של המתודות, האחריות שלכם/ן היא להוסיף `const` ולקבל פרמטרים כ-`reference` היכן שצריך.
- בחלק מהמחלקות תצטרכו להוסיף את שדות המחלקה.
- הקפידו להשתמש במילה `override` בכל מתודה של מחלקה יורשת שדורסת את מתודת מחלקת האב.
- זכרו שמפרק המחלקה הוא מתודה, וגם אליה צריך להתייחס כשמשתמשים בפולימורפיזם.
- חשוב לנסות לממש כמה שיותר קוד במחלקות האב על מנת לחסוך את הקוד מהמחלקות היורשות.
 - לדוגמה, חישבו האם יש מתודות שאפשר לממש במחלקת `Polygon` במקום לממש אותן גם ב-`Triangle` וגם ב-`Rectangle`

שלב 3: הוספת תפריט


שלב 4 שלב בונוס	שלב 3 הוספת תפריט	שלב 2 מימוש המחלקות	שלב 1 כתיבת שלד
--------------------	----------------------	------------------------	--------------------

בחלק האחרון של התרגיל יש לממש תפריט המאפשר למשתמש לבצע 4 פעולות:

1. הוספת צורה חדשה

- יש לתמוך בהוספת אחת מ-4 הצורות (חץ, מעגל, משולש, מלבן)
- יש לקלוט את שם הצורה מן המשתמש.
- יש לאחסן את הצורות במבנה נתונים מסוג `std::vector`
- יש לצייר את הצורה.

2. קבלת מידע או שינוי צורה קיימת

- על המשתמש לבחור את הצורה מתוך הצורות הקיימות.
- לאחר בחירת הצורה המשתמש יוכל לבצע את הפעולות הבאות:
 - הזזת הצורה
 - קבלת פרטי הצורה (שם, סוג, היקף, שטח)
 - הסרת הצורה
- שימו 
 - את **הסרת** הצורה (מחיקה) ניתן לבצע ע"י ציור הצורה בצבע שחור, ואז ציור **כל שאר** הצורות מחדש.
 - את **הזזת** הצורה ניתן לבצע ע"י מחיקת הצורה מהמיקום המקורי ואז ציור הצורה במיקום החדש.

3. מחיקת כל הצורות

4. יציאה

שלב בונוס: אתגרים נוספים

שלב 1 כתיבת שלד	שלב 2 מימוש המחלקות	שלב 3 הוספת תפריט	שלב בונוס אתגרים נוספים
--------------------	------------------------	----------------------	----------------------------



השבוע משימת הבונוס מורכבת משלושה חלקים לא קשורים:

1. שיפור ה-Canvas

המחלקה Canvas, מכילה פונקציה יצירת ומחיקה עבור כל צורה, דבר שגורם לשכפול קוד, מספר רב ומיותר של פונקציות ובעיקר קושי בהוספת צורות חדשות. ממשו את המחלקה BetterCanvas על פי UML הבא:

BetterCanvas
- canvas: Canvas
+ draw_shape (shape: Shape, color: Color)
+ clear_shape (shape: Shape)

שימו 🧡: התרשים מכיל רק את המתודות בהרשאות **public**, תוכלו להוסיף מתודות ושדות פרטיים. המחלקה BetterCanvas תכיל שדה פרטי מסוג Canvas (הצייר הישן). כלומר ניתן לראות ב-BetterCanvas כעטיפת API משופצרת למחלקה הקיימת. בבנאי המחלקה BetterCanvas יש לאתחל גם את השדה canvas

רמז למימוש המתודות:

draw_shape ו-clear_shape יבינו את ה- type של האובייקט ע"י קריאה למתודה getType, של המחלקה Shape. לפי התוצאה היא תקרא לפונקציה המתאימה בשדה canvas. לדוגמא: הפונקציה draw_shape, תקרא ל- Shape::getType, אם התוצאה היא circle, נבצע המרה (casting) של shape ל- Circle, ונקרא ל- Canvas::draw_circle עם הפרמטרים המתאימים. השתמשו ב- switch-case במקום שרשרת תנאים.

2. הוספת המחלקה Quadrangle

המחלקה **Quadrangle** מייצגת מרובע כללי, אותו ניתן להגדיר ע"פ 4 נקודות. הוסיפו את המחלקה וממשו אותה בדומה למחלקות המצולעים האחרות.

3. (grep - בונוס קשה במיוחד)

למי שמחפש/ת אתגר, בצעו את תרגיל הבונוס בנושא smart pointers & RAI, את ההוראות והקבצים הדרושים תוכלו למצוא [בתיקית התרגיל -> קבצים לבונוס](#).

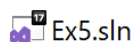
```
C:\Users\000\Documents\Mordehai>"C:\Users\000\source\repos\lightweight_grep\Debug\lightweight_grep.exe" "apple" alice_in_wonderland.txt the_little_prince.txt
=====
Grep Light
By MagShinn
=====
alice_in_wonderland.txt
Found 3 instances of pattern apple in file alice_in_wonderland.txt
-> of mixed flavour of cherry-tart, custard, pine-apple, roast
-> I'm here! Digging for apples, yer honour!
-> 'Digging for apples, indeed!' said the Rabbit angrily. 'Here!
the_little_prince.txt
Found 1 instances of pattern apple in file the_little_prince.txt
-> fBEI am right here,fBX the voice said, fBEunder the apple tree.fBX fBE
C:\Users\000\Documents\Mordehai>
```

! אנא הגישו main נפרד שמכיל שימוש בבונוסים שמימשתם/ן.

נספחים

הגשה ב-GIT

- את הפרויקט יש לנהל ב-Git, לפתוח repository חדש בתוך קבוצת ה-gitlab שלנו ושל המדריך/ה, ולהגיש לינק לפרויקט ב-NEO (אפשר לעשות comment עם הלינק או להגיש מסמך txt עם הלינק בפנים).
- יש להעלות ל-repository את כל הקבצים הרלבנטיים לתרגיל (קבצי txt, מסמכים, ומשאבים אחרים שבהם השתמשנו).
- חשוב להעלות את פרויקט ה-Visual Studio השלם ולהתעלם מקבצים לא נחוצים ([הנחיות במסמך הבא](#)), במידה ולא הועלה הפרויקט השלם, אין להעלות את שאר הקבצים שיוצר Visual Studio – הם רבים מאוד, הם לא מכילים מידע נחוץ להרצת הפרויקט אצל המדריך, ורק יוצרים בלגן.
- הבחירה אילו קבצים להעלות ל-repository נעשית באמצעות הפקודות add ו-rm. אופציה נוספת (מומלצת) היא להוסיף קובץ gitignore. אשר יתעלם מהקבצים הלא נחוצים. במידה ותרצו תוכלו להיעזר ב[סרטוני עזר בנושא GIT](#).
- כסיימתם/ן, בדקו שניתן להריץ את הפרויקט בקלות – בצעו Clone אל תיקייה במחשב אשר שונה מזו שעבדתם/ן, ותראו שהפרויקט נפתח ע"י לחיצה על קובץ ה-sln ויכול לרוץ בלי בעיה



דגשים:

- את הפרויקט יש לפתוח בקבוצת ה-gitlab שאליה משותף/ת המדריך/ה כ-Maintainer.
- יש לוודא שכל הקבצים הרלבנטיים נוספו ל-repository (באמצעות הפקודה add), במידת הצורך ניתן להוריד קבצים מיותרים (באמצעות הפקודה rm).
- יש לבצע commit עבור כל סעיף, ובנקודות שבהן הוספנו שינויים חשובים (לפי הדגשים שהועברו בכיתה).
- עבור כל commit, זכרו לכתוב הודעה קצרה ואינפורמטיבית, שאפשר יהיה להבין מה היה השינוי בקוד.
- יש לדחוף את הקוד (באמצעות הפקודה push) ל-repository בסיום העבודה שלנו, חשוב שבסיום העבודה שלנו, ובמידה ונפנה למדריך/ה, ב-repository יהיה הקוד המעודכן ביותר.
- במידה ושכחנו או שאנחנו לא בטוחים איך מעלים קובץ, או מתעלמים מקבצים, כדאי לצפות בסרטוני ההדרכה בנושא GIT. ניתן לגשת לסרטונים בלשונית ה-resources שבכיתה ה-NEO.
- בסיום העבודה יש להגיש לכיתה ה-NEO קישור ל-repository.

כללי

1. יש לבדוק שכל המטלות מתקמפלות ורצות ב-VS2022. מטלה שלא תעבור קומפילציה אצל הבודק לא תיבדק **והניקוד שלה יהיה 0** 😞
2. יש לבדוק שהקוד שכתבתם עובד. יש להריץ בדיקות שלכם ולוודא שהקוד ברמה טובה.
3. כאשר אתם מתבקשים לממש פונקציה, ממשו בדיוק את הנדרש. אין להוסיף הדפסות וכדו'. אם הוספתם תוך כדי הבדיקות שלכם הדפסות, אנא דאגו להוריד אותם לפני ההגשה.
4. להזכירכם! העבודה היא עצמית, ואין לעשות אותה ביחד.
5. על כל שאלה או בעיה יש לפנות למדריך, לפחות 36 שעות לפני מועד ההגשה.

דגשים לתכנות נכון

- כדאי לקמפל כל מספר שורות קוד ולא לחכות לסוף! הרבה יותר קל לתקן כאשר אין הרבה שגיאות קומפילציה. בנוסף קל יותר להבין מאיפה השגיאות נובעות.
- כדאי לכתוב פונקציה ולבדוק אותה לפני שאתם ממשיכים לפונקציה הבאה. כלומר, כתבו תכנית ראשית שמשתמשת בפונקציה ובודקת האם היא עובדת כראוי. חישבו על מקרי קצה ונסו לראות מה קורה.
- בכל פעם שאתם מתקנים משהו, זכרו שיכול להיות שפגעתם במשהו אחר. לכן עליכם לבדוק שוב מהתחלה.
- חשפו החוצה רק את הממשק המינימלי הדרוש (minimal API), הגדירו את שדות המחלקה כפרטיים, וכמה שפחות מתודות כציבוריות.