OOP

חלק 1 - הקדמה

ביום יום, כולנו משתמשים בעשרות חפצים שאין אנו יודעים כיצד הם בנויים ומורכבים ואין לנו מושג קלוש כיצד הם מבצעים את תפקידם, ולמרות זאת אנו יודעים להשתמש בהם ולנצל את הפונקציונאליות שלהם לטובתנו ותועלתנו.
לדוגמה: מחשב, טלפון סלולארי, נגן MP4, שלט רחוק וכו.'

האם, כמשתמשים, חוסר ההכרות עם הקרביים של אותם המכשירים הוא בעוכרנו?, לא וזו עובדה!
הכיצד זה יתכן?, כיצד ניתן לנהוג במכונית מבלי להכיר את הדרך בה היא ממלאת אחר ייעודה,?
כיצד ניתן להשתמש במחשב מבלי להבין לעומק את צורת הפעולה של המעבד וכיצד הכונן הקשיח כותב וקורא מידע וכיצד ניתן להשתמש בטלפון סלולארי מבלי להכיר את קרביו.?
זה ניתן משום שלמבנה הפנימי ולדרך הפעולה והמימוש הפנימי של אותם המכשירים אין משמעות עבור המשתמש בהם, כל שאנו צריכים להכיר הוא את "ממשק המשתמש" (ורצוי שהוא יהיה ידידותי).

בשביל לנהוג במכונית אנו צריכים לדעת להפעיל את הפונקציונאליות שלה (כיצד להתניע, כיצד לשלב הילוך קדמי/אחורי, כיצד להאט ולעצור, כיצד להדליק את האורות בחשיכה וכדומה) אולם אין אנו צריכים לדעת כיצד המכונית מבצעת וכיצד היא מגיבה לדרישות שלנו ממנה.
בשביל לצפות בטלוויזיה מספיק לדעת כיצד מדליקים אותה, כיצד בוחרים ערוץ, כיצד קובעים את רמת הקול.
אין צורך להבין ולהכיר את מבנה האנטנה וכיצד הטלוויזיה ממירה ומגבירה אותות אלקטרוניים ואנאלוגיים לתמונה וכיצד המכשיר מתמודד עם רעשים בקליטה.
אובייקטים הם "קופסאות שחורות", אנו יודעים מה הם מבצעים, אנו יודעים אלו פעולות ניתן לבצע איתם או עליהם, אולם אין אנו יודעים כיצד הם מבצעים אותם.
וזוהי בדיוק המהות והכוונה של התכונה הראשונה – **Encapsulation** – ניתן להשתמש באובייקטים, אנו יודעים מהם מבצעים, אולם הדרך בה הם מבצעים את הפעולה סמויה ונסתרת מאיתנו.

לדוגמה:
באפליקצית אנדרואיד ישנה מחלקה המטפלת ב-"סל הקניות" של הלקוח. המחלקה "סל קניות" לא פותחה על ידי צוות האתר, אלא נרכשה מחברה המתמחה בכך (אין צורך להמציא את הגלגל מחדש).
המחלקה יודעת לרכז את המוצרים שהלקוח מבקש לרכוש, לסכם את המחיר, לברר את הכתובת למשלוח, לדרוש את פרטי כרטיס האשראי, לשמור את המידע במסד הנתונים, לשלוח אימייל אישור ללקוח, לשלוח את פרטי ההזמנה למחסן וכו.'
התוכניתנים שפיתחו את המערכת ועבדו עם המחלקה, יודעים כיצד להשתמש באובייקט מהמחלקה, יודעים כיצד לחייב את כרטיס האשראי של הלקוח למרות שאין הם מכירים את שירותי הסליקה וכיצד תהליך החיוב מתבצע בפועל. הם יודעים כיצד להשתמש באובייקט מהמחלקה אולם אין יודעים ומכירים את המימושים הפנימיים שלה (הם הרי לא כתבו אותה).
מכירים מצוין את ה-"מה" אולם ה- "איך" הוא חידה גדולה עבורם (וכך גם צריך להיות).

היתרון הראשון של – **Encapsulation** המתכנתים נדרשים להכיר את היכולות של מחלקה אולם לא את המימושים, המתכנתים נדרשים להכיר את מה שהם זקוקים לו אולם יכולים להתעלם מפרטים

מיותרים שיכבידו עליהם.
בעל האתר החליט להחליף את ספק שירותי הסליקה בספק אחר בשל שיקולי עלויות,
האם צריך להחליף את המחלקה?, האם צריך לשנות את הקוד של האתר?, ממש לא,
כל מה שצריך לעשות הוא לשנות את המימוש של המתודה המטפלת בסליקת כרטיסי האשראי.

ומכאן אנו מגיעים ליתרון השני של ,**Encapsulation** יתרון המכונה צמידות נמוכה (Loose Coupling),המימוש של המתודה אינו משפיע על הקוד המשתמש בה, ניתן לשנות את המימוש הפנימי במקרה הצורך (כמו במקרה דנן או כאשר יש באגים) ללא השפעה על הקוד שמשתמש במתודה.

ל Encapsulation -יש יתרונות נוספים (כמו הסתרת מידע למשל) אולם אותם נסקור בהמשך הפרק.
וכיצד מושגת אותה יכולת מופלאה?, כיצד זה בא לידי ביטוי בעולם התכנות?, לזה מוקדש הפרק, אולם לפני כן נערוך הכרות קצרה עם המתודולוגיה עצמה.

חלק 2 - מבוא תיאורטי

שיעור בהיסטוריה במסגרתו נפליג אחורה, לרגע קט, לתחילת שנות ה-60, בתקופה זו הטכנולוגיה מתפתחת בקצב מהיר מבעבר, מחשבים הנם מהירים יותר, קטנים יותר, אמינים יותר וזולים יותר.
ההתפתחות הטכנולוגית המואצת באותם השנים, מובילה למהפכה של ממש, המחשבים שהיו עד אז נחלת משרדים ממשלתיים, ומערכות צבאיות וביטחוניות, נחשקים יותר ויותר על ידי התעשיות האזרחיות.
בתחילת אותו העשור ,התעשיות האזרחיות מתחילות להכיר ביכולת המחשב לשפר תהליכים, להוזילם ולהאיצם, ההתעניינות במחשוב הולכת וגוברת, ומכאן ועד רכישת והטמעת מערכות מחשב הדרך הנה קצרה.

ואכן, מאמצע שנות ה-60 המחשב חודר לחברות תעשיות אזרחיות רבות.
צרכים ביטחוניים שונים מהותית מצרכים אזרחיים, נדרשו תוכנות שיספקו פתרונות לבעיות בהנדסת ייצור, ניהול עובדים, פיתוח ותכנון מוצרים חדשים, פיתרון בעיות אלגוריתמיות מורכבות, סימולאציות לחוזק חומרים וכו. '
השיפור הטכנולוגי הרב, באותן השנים, מאפשר פיתוח מערכות בעלות מורכבות פונקציונאלית גדולה מבעבר, הדרישה לתוכניות מחשב גדולות יותר, מורכבות יותר, מהירות יותר ויעילות יותר הולכת וגוברת.
זהות המשתמשים משתנה מקצה לקצה, בארגונים צבאיים המשתמשים הנם ברובם אנשי מקצוע אשר מכירים את קרביו של המחשב ומתקשרים עמו תוך כדי הקלדת פקודות ב"כתב סתרים" המובן רק להם.
בתעשייה, משתמשי המחשב של תקופה זו יכלו להיות אנשי מקצוע כגון: מהנדסי מכונות, מהנדסי כימיה, פיזיקאים, אנשי אקדמיה וכו'. אנשי מקצוע מעולים בתחומם, אולם ללא כל ידע מקצועי בכל הנוגע לשימוש במחשב. כך שנוספת לה דרישה חדשה, תקשורת "אנושית" וידידותית יותר בין המחשב והתוכנות לבין משתמשי הקצה. הרי לא ניתן לדרוש מאותם כימאים, פיזיקאים, מהנדסים, שבנוסף לבעיות המקצועיות המורכבות אשר עמדו בפתחם, יתמקצעו גם בעולם המורכב והלא מובן של המחשבים דאז, הדרישה הייתה פיתוח פתרונות "אנושיים" יותר.

אולם, דרישה זו, מוסיפה נפח קוד לא מבוטל, רכיבי ממשק אלו יכולים להכיל נפח קוד די מורכב, למרות שלא דובר עדיין על ממשק משתמש גראפי , (GUI) אלא על הוספת יכולת תקשורת פשוטה וקלה בעלת עקומת למידה קצרה ומתונה.

עולם המחשבים, אם כן, נכנס לעידן חדש שבו הדרישות היו גבוהות מבעבר.

מצד אחד האפליקציות היו מורכבות, מסובכות ואנושיות וכפועל יוצא גם גדולות יותר,
ומהצד שני הדרישה הייתה למהירות בפיתוח (עמידה בלוחות זמנים), ולמהירות ויעילות בביצוע (זמן תגובה מהיר, אמינות התוצאות, חיסכון במשאבי חומרה יקרים, יציבות וכו').
וככל שהטכנולוגיה התקדמה, כך גם נדרשה התקדמות רבה במוצרי התוכנה הנלווים.
חברות התוכנה של אותם הימים התקשו לעמוד במשימה, הן לא הצליחו לעמוד בלוחות זמנים, נכשלו בעמידה בתקציבי הפיתוח, התוכנה המוגמרת לא עמדה בציפיות, שגיאות והתרסקויות היו תופעה שכיחה.
על פי אומדנים שונים, בסוף שנות ה- 60 ותחילת שנות ה- 70 הושקעו בארה"ב לבדה כ- 200 מיליארד דולר בשנה בפיתוח אפליקציות מסחריות, אולם רק 5% מהפרויקטים עמדו בציפיות.
בניסיון להשמיש את שאר הפרויקטים, נאלצו חברות הפיתוח לערוך שורה ארוכה של שינויים אשר הובילו לתוצאות כספיות עגומות לאותן החברות. ועדיין, למרות מקצה השיפורים הנוסף רוב הפרויקטים כשלו, רובם נזנחו ולא נכנסו לשימוש מעולם.

ההתפתחות של עולם המחשבים מהעולם הממשלתי/ביטחוני לעולם העסקים לוותה בקשיים רבים ,סכומי עתק ירדו לטמיון, הפסדים כספיים נגרמו לחברות התוכנה וללקוחותיהן, כגודל הציפיות כך גודל האכזבות.
זו התפאורה המלווה את עולם ה"היי טק" של סוף שנות ה- 60 ותחילת שנות ה- 70,
תקופה אשר מכונה "משבר התוכנה.(Software Crisis) "
ובעקבות כך, חברות רבות זנחו (או דחו לזמנים טובים יותר) את רעיון המחשוב ,
חברות נוספות חדלו להאמין שהמחשב מסוגל להתמודד עם הבעיות היומיומיות שלהן.

מנגד, התעוררה חשיבה שונה. במרכזה הטענה שיתכן וניתן יהיה להתמודד עם הקשיים והתקלות באמצעות טכניקות פיתוח אחרות מהמקובל באותה תקופה.
הטכניקה המקובלת באותם ימים הייתה התכנות הפרוצדוראלי .

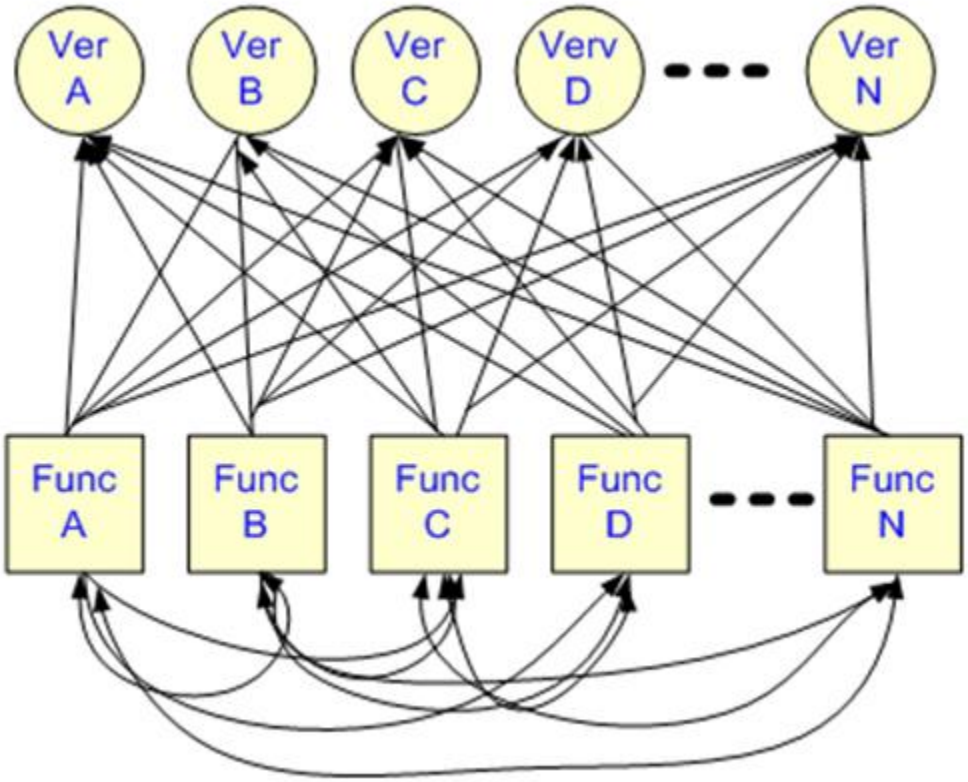
תכנות פרוצדוראלי

בתכנות פרוצדוראלי קיימת הפרדה בין המידע לבין הפונקציונאליות של התוכנית, הדגש בטכניקה זו מושם על החלוקה לפונקציות, המידע מועבר מפונקציה לפונקציה עד לקבלת הפלט.

תפקיד הפונקציות הנו לבצע פעולות על המידע, פונקציה הנה יחידת ביצוע בעלת תפקיד מוגדר. כאשר התפקיד המוגדר מורכב, מחולקת הפונקציה לתתי פונקציות, ואם גם הן מורכבות (מה שבדרך כלל קורה) , אז גם הפונקציונאליות של אותן תתי הפונקציות מחולקת בין תת תתי פונקציות וחוזר חלילה, עד שמגיעים לאוסף של פונקציות "אטומיות", דהיינו, פונקציות המבצעות פעולה בודדה ופשוטה.
מה קורה למידע בתהליך זה, ?
המידע מועבר מפונקציה לפונקציה, אשר בתורה מעבירה אותו לתתי הפונקציות אשר היא מפעילה, וכל אחת מאותן תתי פונקציות מעבירה ..... וחוזר חלילה.
נשמע מסובך? חישבו על תוכניות המכילות מאות פונקציות, אלפי פונקציות, ואף יותר.
יכולת השליטה הולכת ופוחתת.
המידע אינו מרוכז במקום כלשהו, המידע אינו שייך לאיזו שהיא ישות מוגדרת בתוכנית, לעיתים קרובות, מרוב עומס, נשכח מהו התפקיד של משתנה זה או אחר, במקרים אחרים נשכח אף קיומו של אותו משתנה (ויתכן והוא יוגדר מחדש,ונקבל כפילות מידע בתוכנית).
כאשר נקבל שגיאה לוגית) נניח , (age = -17 וננסה לחפש את המקור לשגיאה נמצא את עצמנו צוללים לתוך מספר רב של פונקציות ובודקים את תוכנן. תיקון המקור לתקלה, לאחר שהוא כבר אותר (במידה ואותר) יכול לגרור שגיאות נוספות.

המבנה של טכניקת תכנות זו מותאם למבנה המחשב ולטכניקה בה הוא פועל, אולם טכניקה זו הנה זרה לחשיבה האנושית.

במידה ואנסה לתאר מבנה תוכנית פרוצדוראלית באמצעות איור, לא מן הנמנע שהיא תתואר כמו באיור הבא :



האיור מציג תוכנית המכילה N משתנים ו N -פונקציות. כל אחת מהפונקציות יכולה לגשת לכל אחד מהמשתנים ולקרוא לכל אחת מהפונקציות האחרות בתוכנית, כולם נגישים לכולם ,כולם קריאים לכולם.

מצד אחד ניתן לברך על הגמישות שטכניקה זו מספקת למפתחים, מצד שני כאשר ננסה לתחזק אפליקציה כזו נכיר את צידה האפל של הגמישות המוצעת בה.
הצד האפל מתבטא בקשים רבים בשימוש חוזר בקוד כתוב ,(Reusability) במציאת ותיקון באגים ובהרחבת הקוד הקיים.
במהלך השנים שופרה המתודולוגיה הפרוצדוראלית, שפות פרוצדוראליות רבות הוסיפו מודולים או ספריות, כל מודול התמחה בטיפול בממד מוגדר של התוכנית.
תחילה המדד לחלוקה היה תפקיד הפונקציה, בשנים האחרונות המודולים מורכבים ממבנים ומפונקציות המטפלות באותו המבנה.
החלוקה למודולים שיפרה את קריאות התוכנית, קל יותר למצוא באגים וקל יותר לנצל מודולים קיימים בפרויקטים חדשים .
המבנה המודולארי אמנם מקל, אולם עדיין המלאכה אינה קלה ופשוטה .

Object Oriented Programming

במהלך שנות ה-70 החלה להתפתח מתודולוגית פיתוח חדשה אשר ניסתה לתת מענה לבעיות אשר הובילו את עולם המחשבים ל"משבר התוכנה" . בבסיסה של מתודולוגיה זו עמדה ההכרה שיש לשנות את התפיסה שמפתחים צריכים להתאים את עצמם לדרך הפעולה של המחשב.

ראשוני המתודולוגים של OOP טענו שיש להתאים את טכניקת הפיתוח לצורת החשיבה האנושית. כך יהיה קל וטבעי יותר עבור האדם לפתח אפליקציות מסחריות בהיקף גדול.
הטענה הייתה שיש להתאים את המכונות לאדם ולא לשנות את האדם לטובת המכונות.

ומהם מרכיבי התפיסה האנושית? , על פי אותם חלוצים החשיבה האנושית מתבססת על עצמים (Object, עט, מחשב, עץ, אדם, מזלג וכו ,('מושגים מופשטים (ציון, תפקיד, רעיון, פגישה, הסכם, חוזה וכו'), ושיוכים (סטריאוטיפים, שיוך אובייקט לקטגוריה המספקת אינפורמציה ראשונית ובסיסית על מהותו ותפקידו של האובייקט, לדוגמה: אופניים, מכוניות, מטוסים, רכבות הנם כלי תחבורה. תפוז, אגס, תפוח, בננה הנם פירות וכו').
כאשר אנו מתארים אובייקט במציאות איננו מסתפקים אך ורק בתיאור הפיזי שלו (כיצד הוא נראה, כמה הוא שוקל, מהם מידותיו, מי היצרן אשר בנה אותו וכו') משום שהוא אינו מספק, לכן בנוסף אנו משייכים לאובייקט פעולות (נסיעה, רכיבה, כתיבה, הדפסה וכו').
בצורה זו קל לאדם להתמודד עם כמות המידע האדירה המציפה אותו כל העת, בצורה זו האינפורמציה מאורגנת ומסודרת במוחנו וניתנת לשליפה במהירות כאשר אנו זקוקים לה.
למען האמת, ללא שיטת חשיבה זו יתכן והתקשורת האנושית לא יכלה הייתה להתקיים, כאשר אדם מספר לבן שיחו על מחשב חדש שהוא רכש, אין צורך לתאר ולפרט פרטים שהם ברורים מראש, בן השיח יודע מהו מחשב, הוא יודע שמחשב מכיל מעבד, זיכרון HD ,וכו .'

בבסיסה של התפיסה מונחית האובייקטים עומד רעיון זהה, תהליך הפיתוח הנו בניית "עולם מושגים" מבוסס אובייקטים, מושגים, ושיוכים.
על מנת לאפשר את בניית "עולם המושגים" המתואר למעלה ,כל שפה מונחית אובייקטים חייבת לתמוך בשלושה מרכיבים מרכזיים:

חלק 3 - המחלקה

המחלקה(Class) מחלקות (Classes) הן "אבני הבניין" הבסיסיות בשפות מונחות עצמים,
מחלקה הנה ישות לוגית מופשטת המהווה בית יוצר לאובייקטים ,
ניתן לראותה כתבנית (Template) או כטיפוס ממנה נוצרים האובייקטים.
לדוגמה:

int num1, num2, num3;

המשתנים num1, num2, num3 נוצרים מהטיפוס.int

Person p1, p2, p3;

האובייקטים p1, p2, p3 נוצרים מהמחלקהPerson

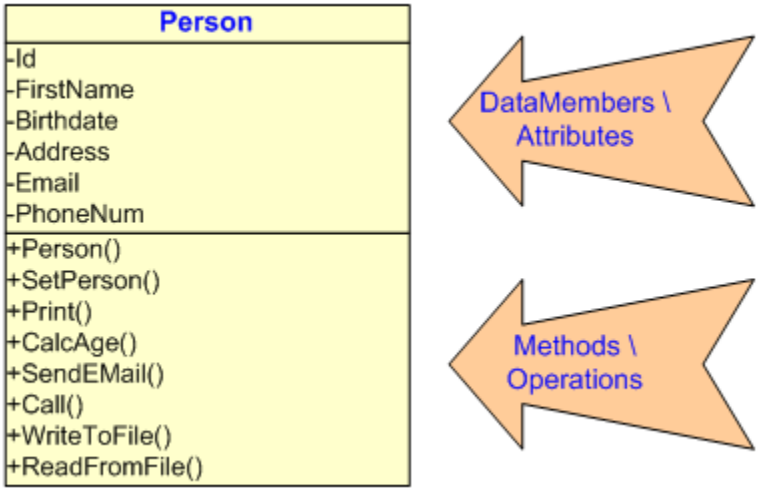
מחלקה מכילה שני מרכיבים מרכזיים : משתנים באמצעותם נתאר אובייקט מהמחלקה ופונקציות אשר מתארות את ההתנהגויות (פונקציונאליות) של האובייקט.
מבנה המחלקה מאפשר את ה ,Encapsulation -באמצעותה אנו מקבלים את היכולת להשתמש באובייקטים מבלי לדעת כיצד הם מבצעים את הפעולות שאנו דורשים מהם (זוכרים את המכונית?, הטלוויזיה?), השימוש באובייקט מממש את התפיסה הנדרשת של "אני יודע מה, לא יודע איך" (ולרוב, גם לא מעניין אותנו).

המחלקה היא ישות לוגית מכיוון שהיא מכילה הגדרות בלבד, המחלקה מגדירה משתנים ומגדירה פונקציות, למעשה היא מגדירה את המסגרת בה מתקיים האובייקט, את תחום עיסוקו ואת תחום

אחריותו.

המחלקה היא מופשטת מכיוון שהיא מתמקדת ומגדירה אך ורק את המשתנים המרכזיים ואת הפונקציות החשובות של האובייקט הנוצר מהמחלקה ומתעלמת מכל אלו אשר אינם חשובים,
במידה והיינו מעוניינים בתיאור מלא של מכונית היינו נדרשים להגדיר אלפי ועשרות אלפי תכונות....
המחלקה הנה "קופסא שחורה" המספקת למשתמשים בה התנהגויות אולם מסתירה מהם את המבנה הפנימי שלה.
המשמעות של "קופסא שחורה" היא שאין שום צורך להכיר ולהבין את המבנה הפנימי של המחלקה על מנת להשתמש בה, כמו שנהג אינו צריך להכיר את מבנה המנוע, ולהיכנס לפרטי הפרטים של מבנה מערכת התמסורת על מנת לנהוג ברכב.
ניתן לומר שמכונית חושפת התנהגויות (האצה, האטה, התנעה, הדלקת/כיבוי אורות וכו'), הנהג יכול לנצל את ההתנהגויות הללו על פי צרכיו, אולם אין הוא יודע ואין הוא צריך לדעת כיצד הן מתבצעות.

מחלקה, מכילה משתנים וגם פונקציות, האיור הבא מתאר מבנה של מחלקה טיפוסית:



משתני המחלקה מכונים גם מאפיינים או תכונות (Data Members) או, (Attributes)
הפונקציות השייכות למחלקה מכונות מתודות או שיטות (Methods) או.(Operations)
האיור מציג מבנה של מחלקה טיפוסית, במחלקה מוגדרות תכונות המתארות אובייקט מהמחלקה,
התכונות מוגדרות כפרטיות באמצעות הסימן (–) משמאל לשם התכונה.
במחלקה מוגדרים גם מתודות (Methods) המתארות את הפעולות שניתן לעשות עם האובייקטים,
המתודות מוגדרות כציבוריות באמצעות הסימן (+) משמאל לשם המתודה.

היכולת של מחלקה להגדיר גם את תיאור האובייקט וגם את הפונקציונאליות שלו היא חשובה ומרכזית בעולם ה OOP -ובעלת ערך מוסף גבוה.
יכולת זו מאפשרת להסתיר את המידע מהעולם החיצון .(Data Hiding)
המשמעות של הסתרת המידע (Data Hiding) מהעולם החיצון היא, שמחלקות אינן יכולות לגשת למידע השייך למחלקה אחרת, הגישה למשתני המחלקה נחסמת הן לצורך קריאה והן לצורך כתיבה,
האלמנטים היחידים אשר יכולים לגשת למידע השייך לאובייקט הן המתודות של אותה המחלקה, והן בלבד.
לכל חבר מחלקה Data Members) או (Methods מוגדרת הרשאת גישה משלו, קיימות מספר הרשאות גישה, השכיחות ביותר הן Public :ו.Private -

Private: חברי מחלקה המוגדרים כ Private -הנם פרטיים למחלקה ולכן מוכרים רק על ידי המתודות של המחלקה, מחלקות אחרות אינן יכולות לגשת אליהם לא לצורך קריאת תוכנם ולא לצורך שינוי תוכנם, המחלקות השונות בפרויקט אף לא מודעות לקיומם של הרכיבים הפרטיים בתוך המחלקה.

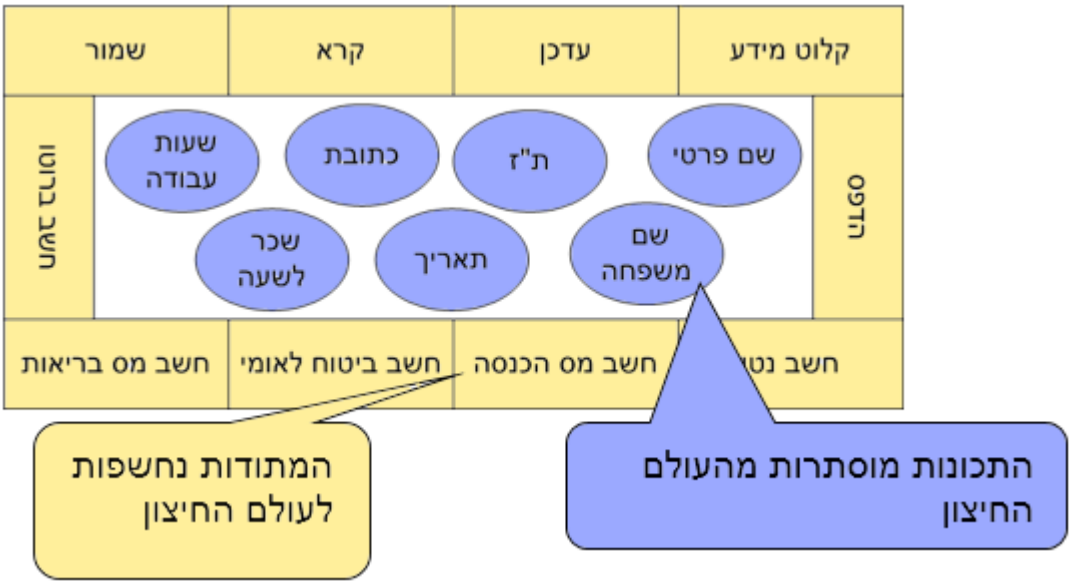
Public: חברי מחלקה המוגדרים כ Public -הנם ציבוריים ולכן הם מוכרים על ידי המחלקות האחרות בפרויקט אשר יכולות לגשת אליהם.

על מנת להגן על הנתונים נקפיד על הכלל הבא:

תכונות (Data Members) תמיד יוגדרו כ.Private -

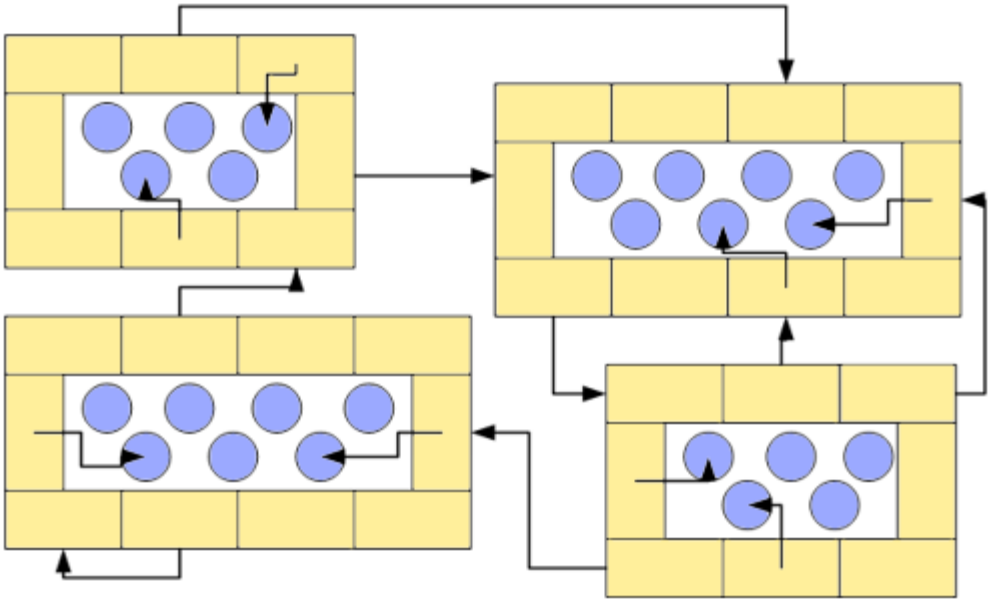
רוב המתודות יוגדרו כציבוריות, אולם ,בעת הצורך, ניתן להגדירן גם כפרטיות.

הערך המוסף של כימוס (Encapsulation) בא לידי ביטוי הן באיתור ותיקון שגיאות לוגיות והן כאשר נוצר צורך לשנות את המבנה הפנימי של המחלקה.



האובייקטים מסתירים את התכונות (Data Members) אולם חושפים פעולות , (Methods) אובייקט חיצוני המעוניין במידע מתוך מחלקה מסוימת חייב לקרוא למתודות של אותה המחלקה על מנת לקבל מהן מידע, אין לו גישה ישירה אליהן.

האיור הבא מדגים זאת:



כל ישות הפעילה בתוכנית מיוצגת באמצעות מחלקה, הישות יכולה להיות ישות מוחשית (מכונית, עובד, מחשב, ספר וכו'),
ישות יכולה להיות ישות מופשטת (תיאור פגישה, ציון, חוזה, תפקיד וכו'),
ישות יכולה לייצג אירוע (פגישה, טיסה, מסיבה, הלוואה, נסיעה, השאלה וכו')

מחלקה, כאמור הנה ישות לוגית או תבנית, בית היוצר לאובייקטים.
אובייקט הנו ישות פיזית, מופע (Instance) של מחלקה, ממחלקה אחת ניתן להגדיר הרבה מאוד אובייקטים, לדוגמה:
המחלקה Circle מגדירה את התכונות הבאות:
צבע , (Color) גודל (Width, Height), מיקום.(X,Y)
בתוכנית מחשב נוצרו האובייקטים הבאים ממחלקה זו:
הראשון צבעו ירוק, גודלו Width= 100 Height=100, ומיקומו.x=10, y=30
השני צבעו כחול, גודלו Width= 120 Height=120, ומיקומו, x=20, y=150
השלישי צבעו אדום, גודלו Width= 70 Height=70, ומיקומו. x=200, y=230

מהמחלקה Circle הוגדרו שלושה אובייקטים שונים (והיד עוד נטויה...)

גריידי בוך ,(Grady Booch) אחד המתודולוגים המובילים בעולם ה OOP -קבע את ההגדרה הבאה:
"לכל אובייקט יש מצב ,(State) התנהגות (Behavior) וזהות ייחודית".(Identity)
– **State**בכל מחלקה מוגדרות תכונות (Data Members) המתארות את האובייקט, ניתן גם לראותן כמבנה הנתונים של המחלקה.
כל אובייקט אשר נוצר ממחלקה מסוימת מכיל את אותו מבנה הנתונים. בכל זמן נתון יש לכל אחת מהתכונות הללו ערך.(Value)
אוסף הערכים הללו (או חלקו) מייצג את ה State -של האובייקט, ה State -הוא דינאמי ויכול להשתנות על ציר הזמן.
ה State -מייצג מצב זמני בו נמצא האובייקט. ה State-ישתנה כתוצאה בהפעלת מתודות.
לדוגמה : תכונה של מכונית המייצגת את מהירות המכונית מכילה את הערך 80 ,(Speed=80) ולכן המכונית נמצאת במצב "נסיעה", התכונה Speed מכילה את הערך 80 כתוצאה מהפעולות "סע" או

"האץ" שהופעלו על האובייקט. כתוצאה מפעולת "עצור" התכונה המייצגת את המהירות תתאפס (Speed=0)ולכן המכונית משנה את מצבה ל"עצירה."

יתכן ותכונות נוספות ישפיעו על המצב של האובייקט (סל"ד, הילוך וכו,) ויתכן ויהיו תכונות אשר לא ישפיעו על המצב הנוכחי בו נמצא האובייקט (כגון: צבע המכונית), אולם הן יכולות להשפיע על מצבים אופציונאליים אחרים של האובייקט.

- **Behavior**כל מחלקה מגדירה מתודות, כל מתודה מייצגת פעולה מוגדרת שהאובייקט יכול לבצע.
ההתנהגות של אובייקט מוגדרת כאוסף כל הפעילויות שהאובייקט יודע לבצע.
מכיוון שהמתודות מוגדרות במחלקה כל האובייקטים מאותה המחלקה מכילים התנהגות זהה.
ההתנהגות משפיעה על המצב (State) בו נמצא האובייקט.

– **Identity**לכל אובייקט יש מזהה ייחודי המבדיל בינו לבין שאר האובייקטים מאותה המחלקה.
הזיהוי הינו קבוע ואסור שהוא יושפע ממצב (State) המחלקה.
לדוגמה:
יתכן מצב ששתי מכוניות זהות (אותו היצרן,אותו הדגם, אותו המודל ואותו הצבע) נמצאות במצב זהה (נסיעה במהירות של 80 קמ"ש), ניתן יהיה להבחין ביניהן באמצעות מספר הרכב המציין את ה - Identityשל כל מכונית.
כאשר המכוניות ישנו את מצבן לעצירה, מספר הרכב אינו משתנה.
במקרה של דוגמה זו לא ניתן להתייחס לאחת המכוניות על פי היצרן, הדגם או המודל או הצבע, לא ניתן להתייחס אליהן על פי המצב הזמני בו הן נמצאות, ניתן להתייחס לאחת מהן רק על פי מספר הרכב .
באמצעות משתני המחלקה המייצגים את הזיהוי ניתן לבודד את האובייקט הרצוי ולהתייחס רק אליו.

הגדרת מחלקה בסיסית

בתוכניות בשפת java מוגדרות לא מעט מחלקות, לכל מחלקה תפקיד מוגדר ותחום אחריות משלה, מקובל שכל מחלקה מוגדרת בקובץ נפרד הנושא את שם המחלקה.
אין זו חובה מבחינת השפה, אלא זהו הרגל תכנותי נכון המאפשר להתמצא בקלות במבנה תוכניות גדולות.

תחביר:

האובייקט

אובייקט הוא מופע (Instance) של מחלקה, ממחלקה ניתן להגדיר הרבה אובייקטים (או מופעים.(
מחלקה הנה מבנה לוגי/הגדרתי/הצהרתי/מופשט, אובייקט הינו ישות פיזית בעל אורך חיים (Life time)

וטווח הכרה.(Scope)
המחלקה היא "בית יוצר" לאובייקטים, ניתן לראותה גם כתבנית ממנה נוצרים אובייקטים בעלי מבנה זהה, אולם בעלי ערכים שונים.

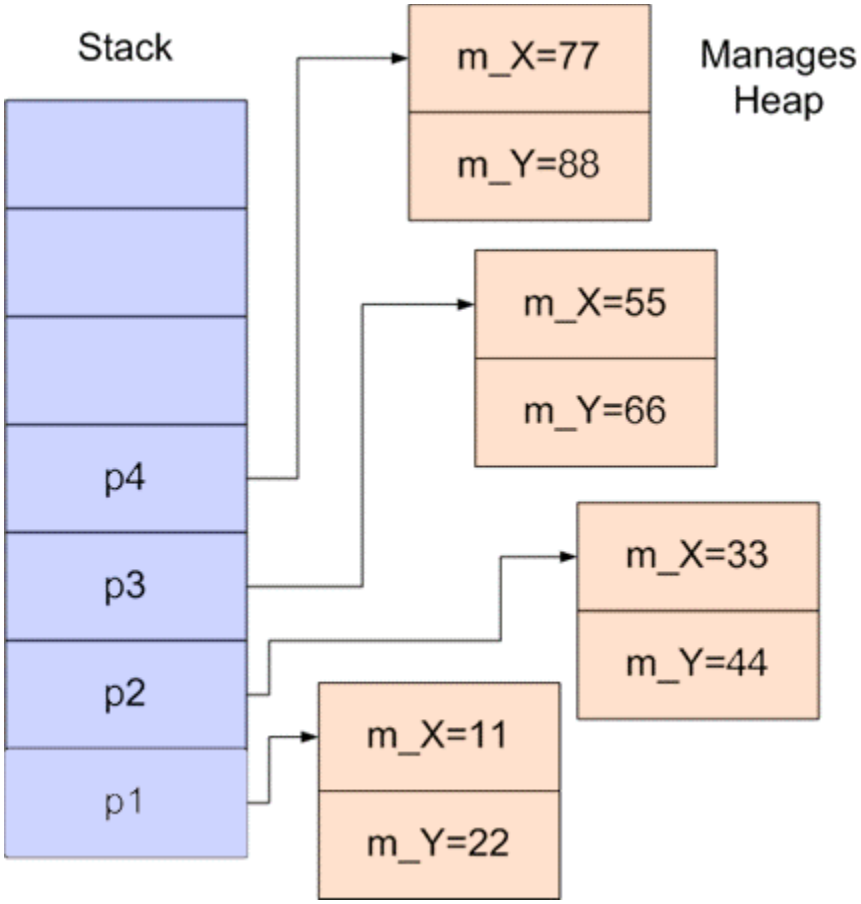
יצירת אובייקט ב java מתבצעת בשני שלבים:

- הגדרת ייחוס (Reference) מטיפוס המחלקה, ייחוס זה נמצא באזור זיכרון הנקרא Stack.
- הקצאת זיכרון לאובייקט והעברת כתובתו לייחוס. האובייקט מוקצה ב- Managed Heap.

הקצאת האובייקט מתבצעת באמצעות האופרטור new.
לדוגמה:

```
Point p = new Point();
```

תמונת זיכרון:

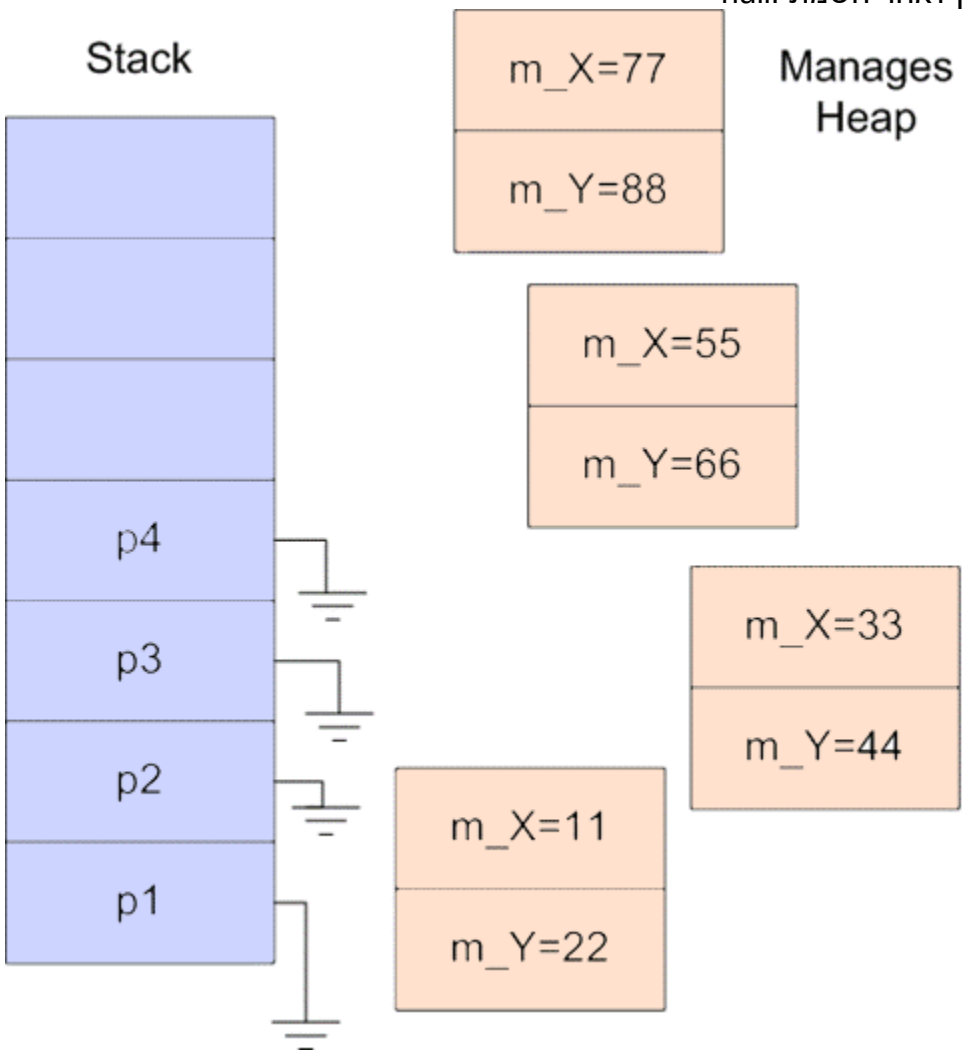


בשונה מהטיפוסים הפשוטים (int, float, double, short וכדומה) ובדומה למערכים כאשר נוצר אובייקט התכונות שלו מאופסות.

כאשר התוכנית אינה זקוקה יותר לאובייקט מומלץ לנתק את הקשר בין האובייקט לייחוס על ידי השמת הערך nullבייחוס.
השמת null בייחוס אינה משחררת את האובייקט, אולם היא תאיץ, במידת הצורך, את שחרור הזיכרון שלו על ידי ה.Garbage Collection -
לדוגמה:

```
p = null;
```

תמונת זיכרון לאחר השמת null:



מתודות

אם היינו מתבקשים לכתוב מחלקה המייצגת טוש סביר להניח שהיינו מגדירים אותה כדלקמן:
תכונות: שם יצרן, דגם, מחיר, צבע, עובי, כמות דיו וכו.'
פעולות: צייר עיגול, צייר מלבן, צבע שטח, כתוב מחרוזת וכדומה.
שאלה:
איזה חלק המחלקה יותר חשוב: התכונות או המתודות? (התשובה שניהם חשובים לא תתקבל).
תשובה:
המתודות יותר חשובות משום שהן למעשה הערך המוסף של המחלקה, והן הסיבה האמיתית להגדרת אובייקטים מהמחלקה.

במציאות אנו לא נרכוש טוש רק בגלל שהוא במבצע או רק בגלל שהוא בצבע אדום או רק בגלל כמות הדיו, אנו נרכוש טוש כאשר נצטרך לצייר עיגול או לשרטט מלבן או כאשר נרצה לכתוב מחרוזת על לוח.
כך גם בעולם התכנות, אנו נגדיר מחלקה המייצגת טוש ונקצה ממנה אובייקטים כאשר יש משמעות לפונקציונאליות שלה במערכת (יש מקרים שבהם המידע הוא הפונקציונאליות הנדרשת, כמו בדוחות למשל, אבל זה המקרה הפחות שכיח).

אובייקטים "עושים משהוא" בתוכנית, וה"משהוא" שהם עושים משפיע על ההתנהלות ואו התוצאות של התוכנית, אחרת אין בהם צורך.
מחלקות מגדירות לא רק את תכונות המחלקה אלא גם את ההתנהגות של המחלקה, דהיינו, את הפעולות שהמחלקה מסוגלת לבצע ובכך להשפיע על התוכנית.
הפעולות מוגדרות באמצעות המתודות של המחלקה.

בתכנון מחלקה יש לקחת בחשבון תחילה את ההתנהגויות הנדרשות, תכונות המחלקה הנדרשות הן לעיתים תוצאה של צרכי ההתנהגויות.
וזה תפקיד המתודות, ולשם כך הן נדרשות, להגדיר את הפונקציונאליות של האובייקט במערכת .

כתיבה וקריאה

ישנן שני סוגים של מתודות: מתודות סטאטיות ומתודות מופע.
מתודות סטאטיות הן מתודות ברמת המחלקה ולא ברמת האובייקט (לא הגדרנו אובייקט בשביל לקרוא להן).
מתודות סטאטיות הן מתודות שאינן מייצגות התנהגות או פעולה של אובייקט אלא מייצגות שירות שהמחלקה מספקת לתוכנית או לכלל האובייקטים של המחלקה.
בתכנון ובכתיבת מחלקות יש צורך לתכנן ולכתוב מתודות המטפלות באובייקט או מייצגות פעולה שהאובייקט מסוגל לבצע (או שאנו מסוגלים לבצע עליו).
מתודות הללו לא יוגדרו כמתודות סטאטיות אלא כמתודות מופע.(Instance Method)
מתודות מופע מוגדרות בדיוק כמו מתודות סטאטיות למעט הסרה של המילה השמורה .static
מתודות סטאטיות הופעלו על ידי המחלקה, מתודות מופע מופעלות על ידי האובייקט.
כל החוקים, הכללים והמוסכמות שנסקרו בפרק 11 תקפים גם בהגדרת מתודות מופע.
מחלקה יכולה להכיל מתודות מופע לצד מתודות סטאטיות.

בנאים

שפת java מאפשרת להגדיר מתודות מיוחדות הנקראות בנאים Constructors) או actorבקיצור.(
מתודת הבנאי (Ctor) מופעלת פעם אחת בחיי אובייקט, ופעם אחת בלבד, כחלק מתהליך יצירת האובייקט.
מהו תפקיד מתודת הבנאי,?
למתודת הבנאי תפקיד כפול: הקצאת האובייקט ואתחול התכונות.
מאחורי הקלעים ובצורה שקופה למתכנת מתודת הבנאי היא זו שיוצרת את האובייקט,
בנוסף הבנאי מאתחל תכונות המחלקה בערך ראשוני.
בנאי אם כך, תפקידו כפול:
הקצאת האובייקט (שקוף למתכנת), ואתחול תכונות המחלקה.
יתכן והאתחול יאפס את ערכי התכונות או לחילופין יבצע להן השמה על פי ערכים אשר מתקבלים כפרמטרים.
עם סיום פעולתו, הבנאי מחזיר את כתובתו של האובייקט שזה עתה נוצר ואותחל לייחוס.
לכל מחלקה מוגדר לפחות מתודת בנאי אחת, במידה ולא נגדיר מתודת בנאי היא תוגדר בצורה מרומזת

ומעבר ליצירת האובייקט היא תאפס את ערכיו, בנאי זה נקרא בנאי ברירת מחדל.(Default Ctor)

הייחוס this

כל אובייקט מכיל העתק פרטי של התכונות שלו במיקום שונה בזיכרון, מה שכמובן נראה טבעי והגיוני, כי אחרת לא ניתן היה להגדיר ערכים אחרים לתכונות של אובייקטים, אולם המתודות משותפות לכל האובייקטים מאותה המחלקה.
למרות שלא נוצר העתק של המתודות עבור כל אובייקט מהמחלקה, יודעת המתודה בדיוק איזה אובייקט הפעיל אותה, וזאת למרות שמימוש המתודה לא מכיל שום התייחסות מפורשת לאותה המתודה.
לדוגמה:

```
e1.Print();
e2.Print();
```

המתודה Print() תדפיס את המידע של e1 כאשר האובייקט e1 יפעיל אותה , ואת הפרטים של e2 כאשר האובייקט e2 יפעיל אותה , בשני המקרים מופעלת אותה המתודה בדיוק. כיצד הדבר מתרחש?, כיצד המתודה יודעת באמצעות איזה אובייקט היא הופעלה?
כאשר קוראים למתודה, הקומפיילר מוסיף לרשימת הפרמטרים של המתודה פרמטר חשוב נוסף, הפרמטר הנוסף הנו ייחוס לאובייקט שהפעיל את המתודה, הייחוס נשלח למתודה בצורה מרומזת (Implicit), במקרה הראשון כאשר המתודה Print() תופעל, היא תקבל פרמטר יחיד המכיל את הכתובת של האובייקט e1 ובפעם השנייה אותה המתודה Print() תקבל את הכתובת של האובייקט. e2
הייחוס נקרא this, הוא מתווסף אוטומטית לכל המתודות (למעט למתודות סטאטיות) ואם נרצה ונמצא זאת לנכון נוכל אף להשתמש בו.
המילה this הנה מילה שמורה.

השימוש באופרטור this מותר גם בקריאות למתודות אחרות של המחלקה.
כמובן שלא ניתן להשתמש באופרטור this במתודות סטאטיות משום שהן מופעלות על ידי המחלקה ולא על ידי אובייקט ספציפי.

חברי מחלקה סטאטיים

Static Data Member

עד כה הגדרנו תכונות (Data Members) השייכות לאובייקט.
בכל יצירת מופע (Instance) של אובייקט נוצרים בזיכרון) ב (Manage heap -העתקים של אותם המשתנים החברים, לכל אובייקט יש Data Members משל עצמו ,אין שום קשר בין ה Data Members -של אובייקט אחד ל Data members -של האובייקטים האחרים מאותה המחלקה.

במילים אחרות, אם נגדיר מספר אובייקטים מהמחלקה מסויימת, לכל אובייקט יוקצו בזיכרון כל ה Data - Membersהמוגדרים במחלקה, תוכנם של אותם המשתנים יהיה שונה מאובייקט לאובייקט.

לעיתים נרצה לנהל מאגר נתונים משותף לכל האובייקטים הנוצרים ממחלקה מסוימת, הסיבה לכך יכולה להיות חיסכון בזיכרון או ביצוע אלגוריתם מסוים.
לדוגמה, נניח שברצוננו למנות את מספר האובייקטים שנוצרו ממחלקה מסוימת מאז שהתוכנית החלה לרוץ או בתכונה אחרת נשתמש בערך זהה לכל האובייקטים.

אורך החיים של תכונות סטאטיות

מכיוון שתכונות סטאטיות מוגדרות ברמת המחלקה ולא ברמת האובייקט הן נוצרות כאשר המחלקה נטענת לזיכרון עם תחילת ריצת התוכנית (לרוב).
ולכן ניתן לגשת אליהן לפני שנוצר אובייקט מהמחלקה ובלי קשר לקיומו.

Static Method

Static Methodsדומות ל Static Data Members -בכך שהן אינן שייכות לאובייקטים אלא הן ברמת המחלקה .
Static Methodsזמינות מהרגע שהמחלקה נטענת לזיכרון.
כמו Static Data Members גם Static Methods מוגדרות באמצעות המילה השמורה , static תחביר:

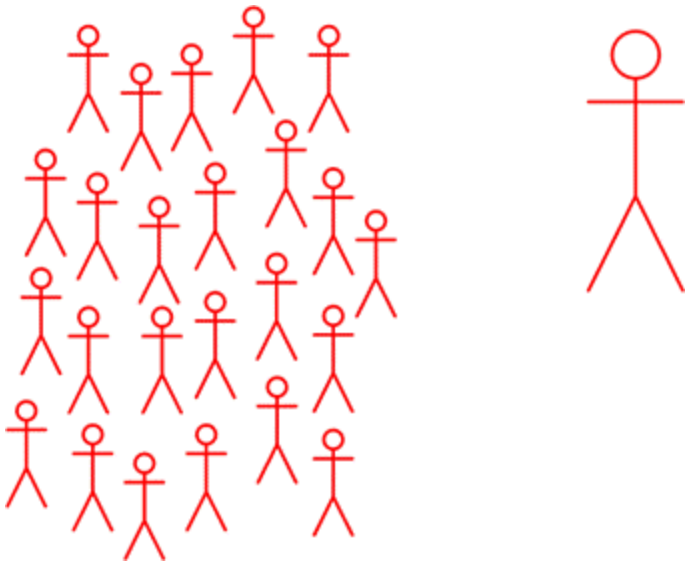
מכיוון שהן משויכות למחלקה ולא למופע (instance) מסוים מהמחלקה, הן מופעלות באמצעות שם המחלקה:

ClassName.StaticMethodName (. . .);

מכיוון ש Static Methods -זמינות מהרגע בו המחלקה נטענת לזיכרון, ולפני הקצאת האובייקט הראשון הן יכולות לטפל אך ורק ב.Static Data Members -
Static Data Membersאינן מקבלות את הייחוס. this

מחלקות ומערכים

הטיפול באובייקט יחיד שונה מטיפול באוסף אובייקטים.



ההתנהגות הנדרשת מאובייקט מהמחלקה Person יכול להיות: הגדרת ערכים לתכונות, בדיקה האם חל יום ההולדת, שליחת אימייל וכו.'
תוכנת ספר טלפונים מטפלת באוסף של אובייקטים מהמחלקה.Person
ההתנהגות הנדרשת מאוסף אובייקטים מהמחלקה Person תהיה שונה: הוסף אובייקט, מחק אובייקט, חפש אובייקט, מיין את רשימת האובייקטים וכו.'

תכנון ועיצוב נכון של מערכות מבוססות OOP מפריד בין המחלקה המטפלת ביחיד לבין המחלקה המטפלת באוסף האובייקטים הקיים במערכת.
לדוגמה:
בתוכנית "ספר טלפונים" נגדיר מחלקה בשם Person אשר תטפל בכל ההיבטים הקשורים באובייקט יחיד ומחלקה נוספת בשם PersonArray אשר תטפל בכל ההיבטים של אוסף האובייקטים מטיפוס Personהקיים בספר הטלפונים.

נגדיר לכל אחת מהמחלקות את המתודות הנדרשות:
במחלקה Person נגדיר את המתודות המטפלות בהתנהגויות של הגדרת ערכים לתכונות, בדיקה האם חל יום ההולדת, שליחת אימייל, עדכון ערכים.
למחלקה PersonArray נגדיר את המתודות המטפלות בהתנהגויות של הוספת אובייקט, מחיקת אובייקט, חיפוש אובייקט, הדפסת שמות האנשים וכו. '

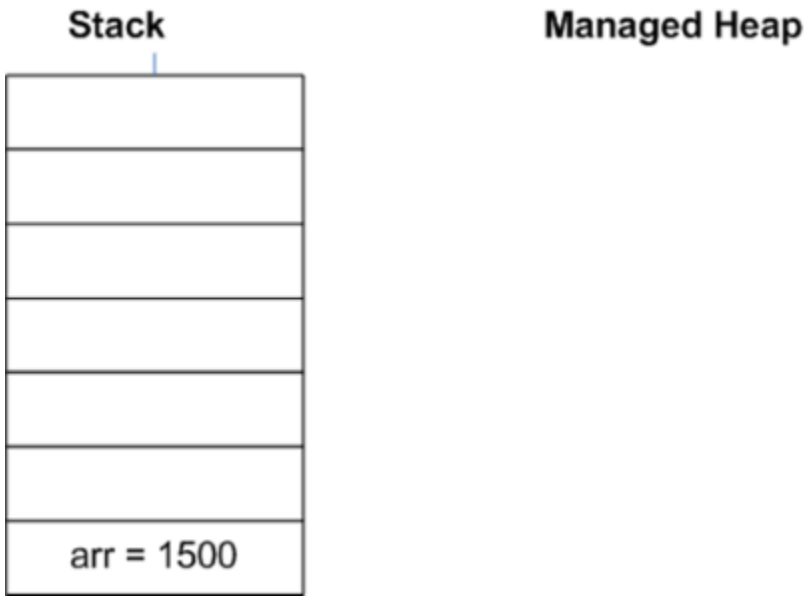
מערך של אובייקטים

אובייקטים הם מטיפוס ערך (Reference Types) ולכן בניית מערך של אובייקטים כרוכה בהוספת שלב נוסף לשלבים המוכרים לנו מהגדרת והקצאת מערך של. Value Types
בניית מערך של אובייקטים מתבצעת בשלושה שלבים:
- 1הגדרת ייחוס למערך.
- 2הקצאת המערך- יוצרת את המערך עצמו המכיל ייחוסים לאובייקטים בלבד.
- 3הקצאת האובייקטים והעברת כתובתם (הייחוס) שלהם למערך.

הגדרת ייחוס למערך של אובייקטים.
בשלב זה קיים ייחוס למערך אבל עדיין מערך:

```
Person[] arr = new Person[5];
```

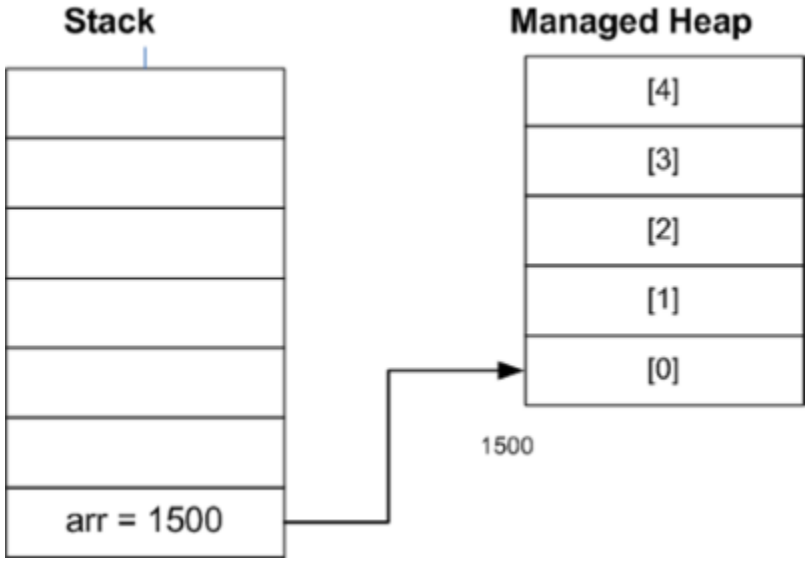
תמונת זכרון:



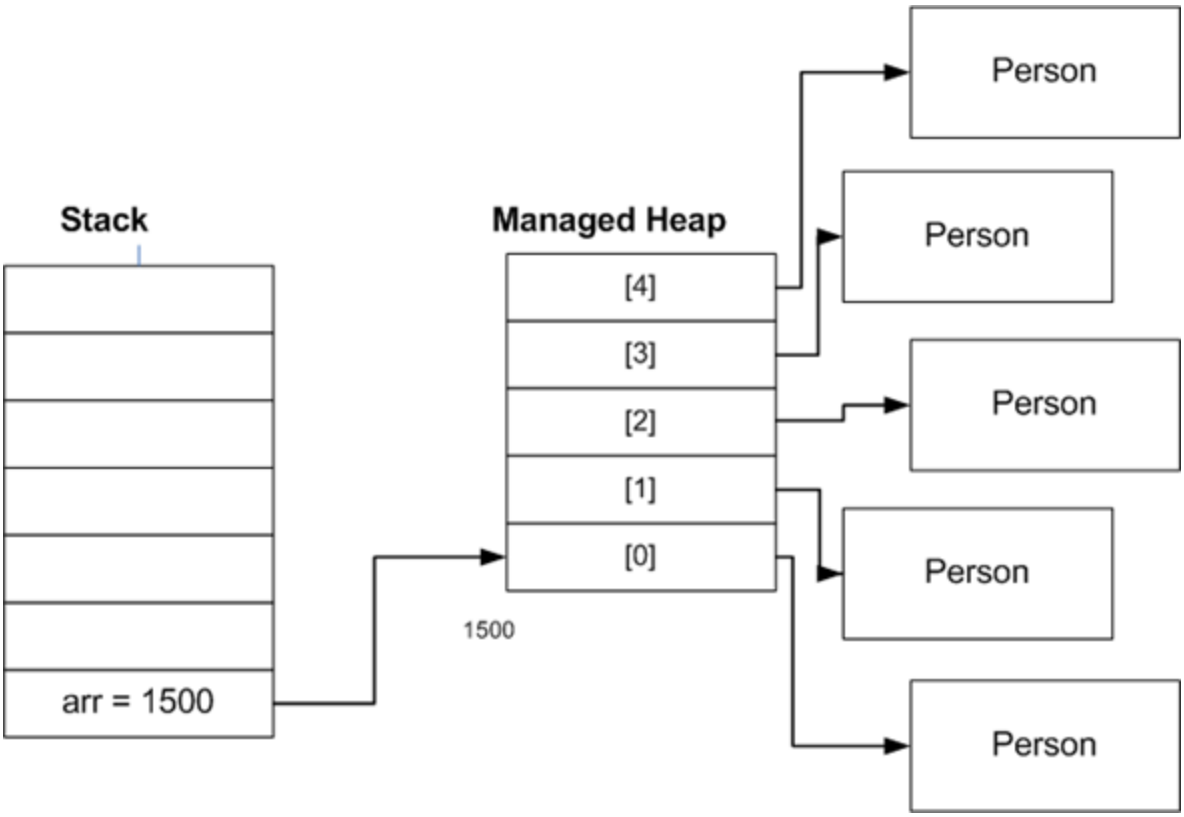
שלב שני, עדיין **בשורה 5** ,הקצאת מערך של ייחוסים לאובייקטים

```
Person[] arr = new Person[5];
```

בשלב זה קיים יחוס למערך, אולם הוא ריק ואין בו אובייקטים.



כל אחד מהאיברים 0-4 הוא ייחוס לאובייקט, אולם בשלב זה הוא אינו מיוחס לשום אובייקט, את זה נבצע בשלב הבא: נקצה את האובייקטים, ונעביר אותם למערך p ,הוא ייחוס לאובייקט מהטיפוס Personואותו נעביר למערך (לאחר הקצאתו), איברי המערך מיוחסים לאובייקט שממוקם אי שם במרחבי הזיכרון.
תמונת הזכרון לאחר שלב זה:


Stack
arr = 1500
Managed Heap
[4]
[3]
[2]
[1]
[0]
1500
Person
Person
Person
Person
Person