פרופ' עמיר תומר
amir@amirtomer.com

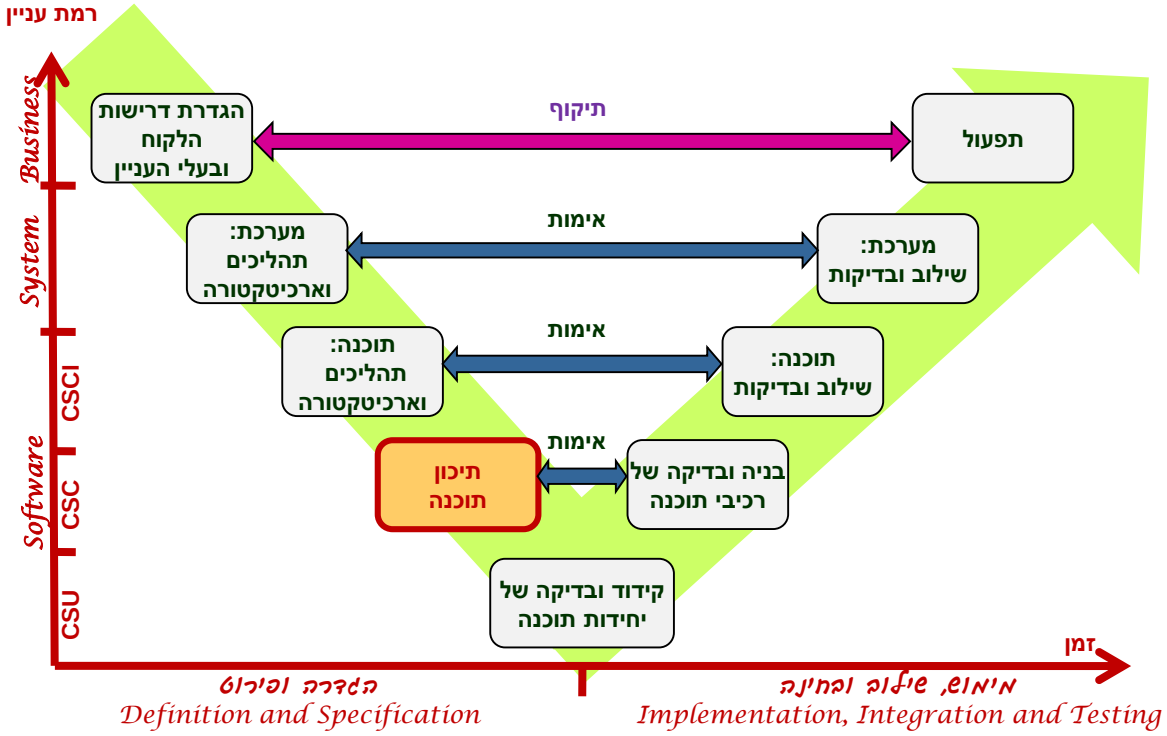
פרק 8

תכן תוכנה מונחה עצמים

Object-Oriented Software Design

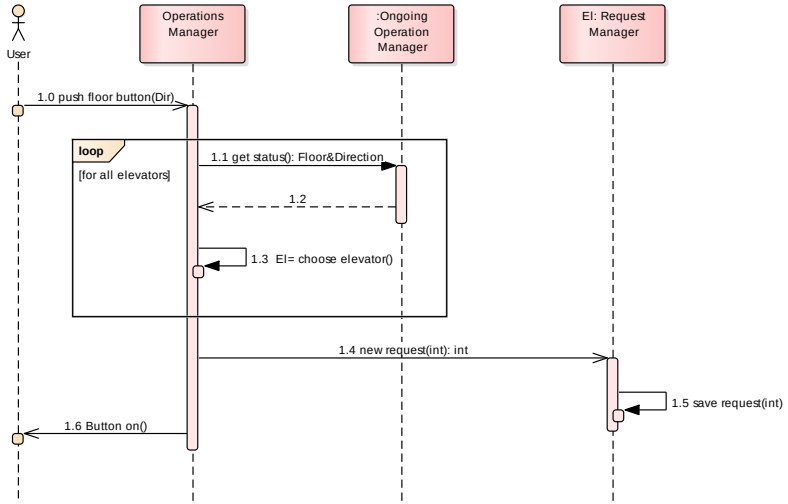
פעילות תיכון התוכנה (בגישה מונחית העצמים)

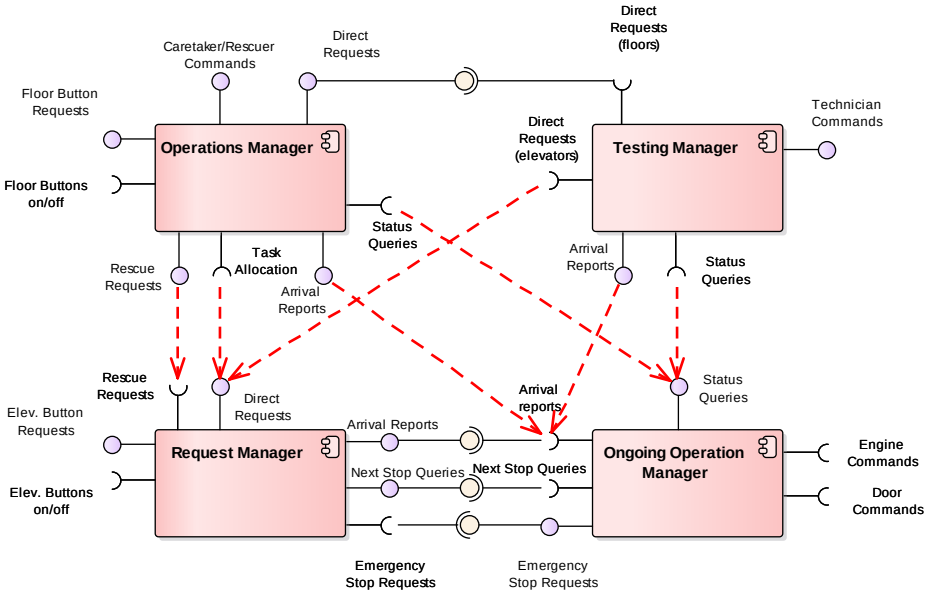
- **מטרת הפעילות**
  - הגדרת המודולים (מחלקות מהן יווצרו עצמים) הבונים את התוכנה
  - הקצאת פונקציונאליות למחלקות (מאפיינים ומתודות)
- **קלט**
  - ארכיטקטורת התוכנה (Component Diagram)
  - תהליכי התוכנה (Sequence Diagrams) ברמת רכיבים
- **תוצרים**
  - מודל מחלקות (Class Diagram)
  - תהליכי תוכנה (Sequence Diagrams) ברמת עצמים



תמונת המצב עד כה

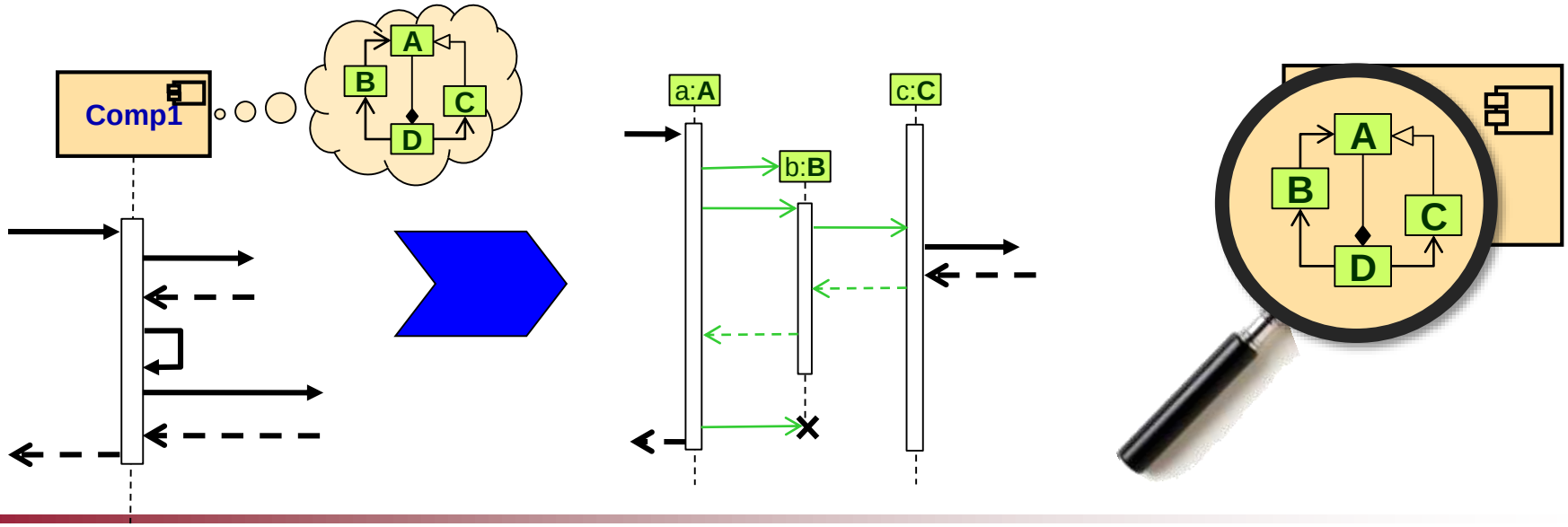
- בנינו ארכיטקטורה של מערכת התוכנה, המפרטת את הפירוק המערכתי של כל פריט תוכנה (ברמת המערכת)
  - מטרות: ביצוע תהליכי המערכת (Use Cases)
  - רכיבים: Software Components עם תפקידים וממשקים
  - מבנה: הקשרים הפנימיים והחיצוניים באמצעות הממשקים (component diagram)
  - התנהגות: אינטראקציה בין הרכיבים למימוש התהליכים (sequence diagrams)
- ארכיטקטורה התוכנה מאפשרת גם לתאר את האינטראקציה בין פריטי התוכנה השונים



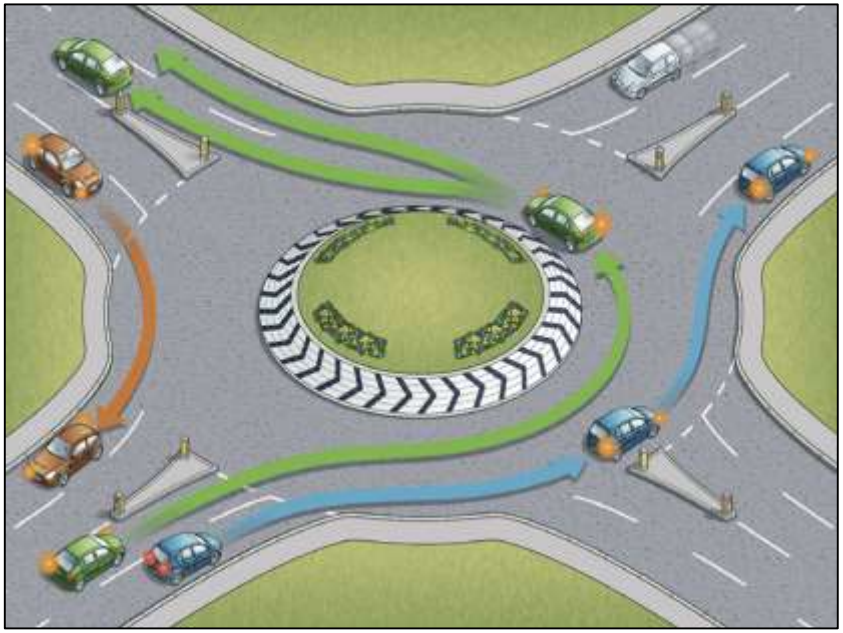


תכן תוכנה (מונחה עצמים)

- ברמה הבאה בפירוק המערכתי עלינו לפרט עבור כל רכיב (software component):
  - מטרות : מימוש הפונקציונאליות של הרכיב
  - מרכיבים: מודולי תוכנה (עצמים)
  - מבנה: הקשרים (ההיכרות) בין העצמים השונים
  - התנהגות: האינטראקציה בין העצמים השונים המביאה למימוש הפונקציונאליות

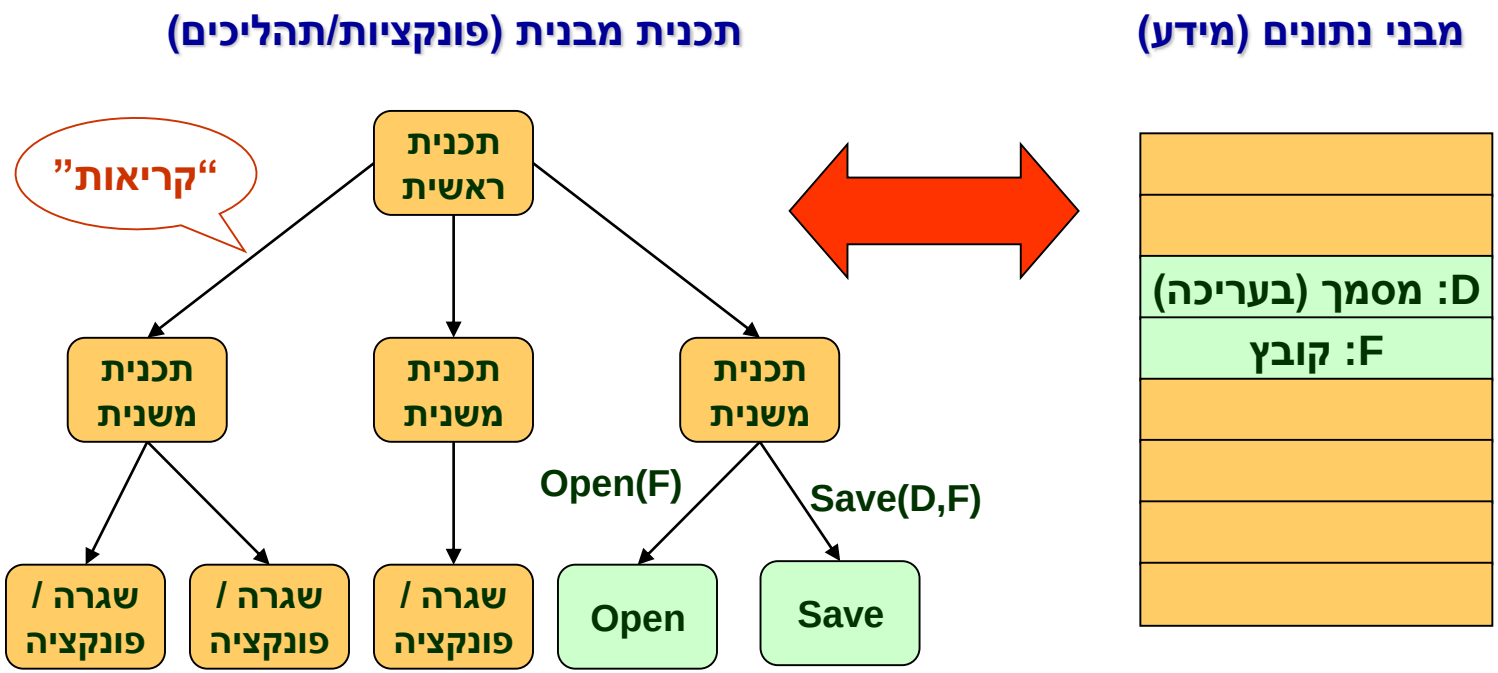


מה ההבדל?



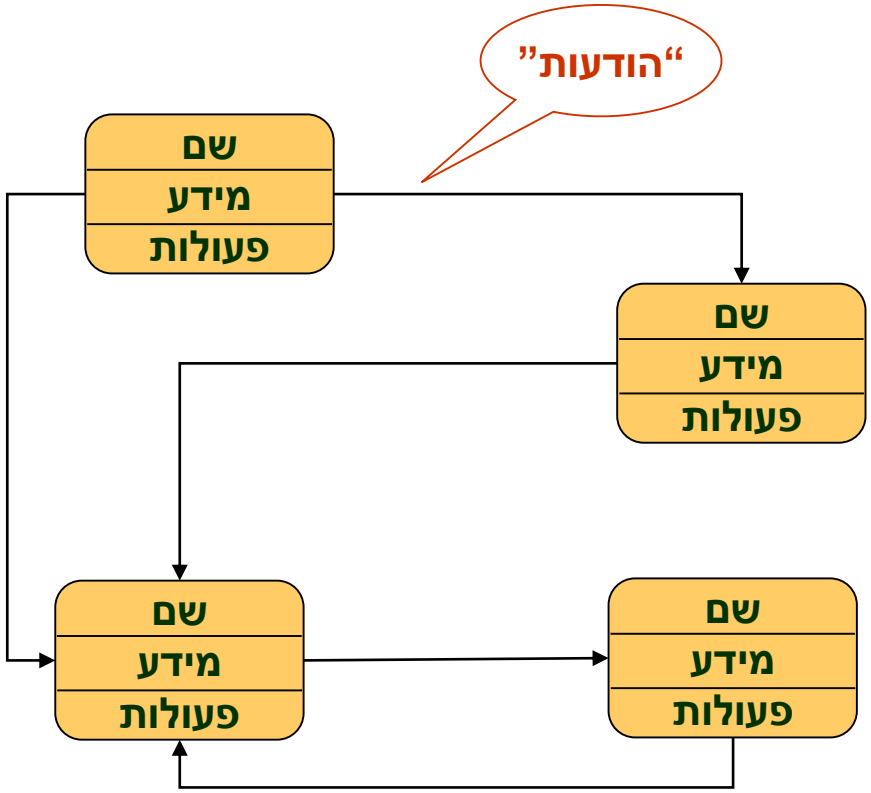
תוכנה מבנית (structured) [קלאסית]

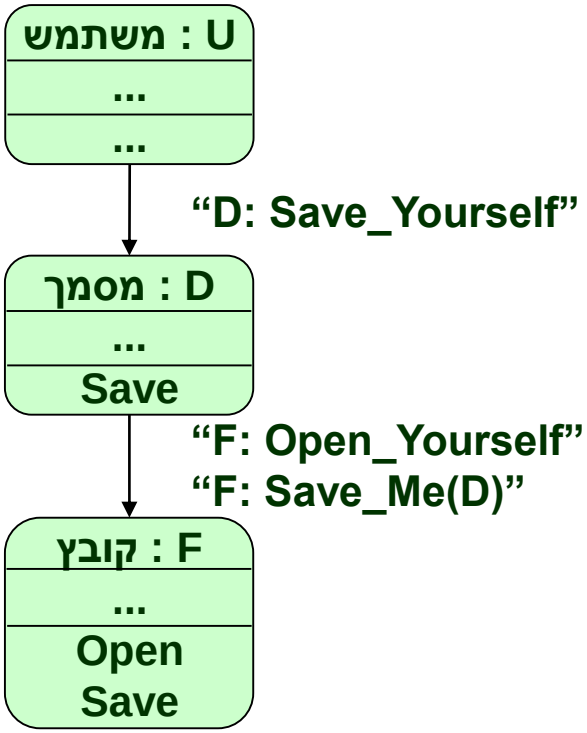
- מבנה
  - הפרדת המידע (data) מהפונקציונליות
- ביצוע
  - "עץ" קריאות בין מודלים
  - כל מודול מבצע מניפולציות על המידע



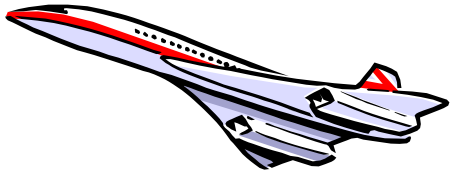
תוכנה מונחית עצמים

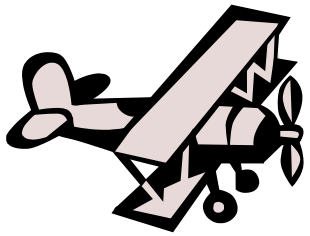
- מבנה
  - המידע והפונקציונליות כמוסים (encapsulated) בתוך "עצמים"
- ביצוע
  - העברת "הודעות" בין אובייקטים

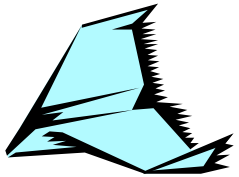




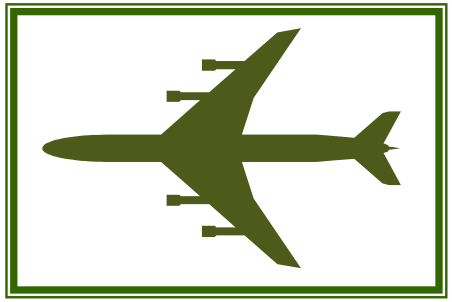
התבנית מונחית העצמים (Object Oriented paradigm) – מושגי יסוד







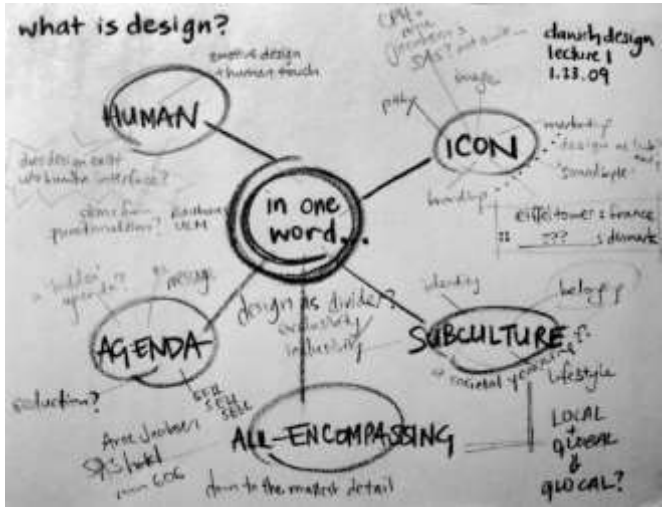
- עצם (object)
  - ישות בדידה
  - גבולות וזיהוי מוגדרים
  - מכיל בתוכו (encapsulates) מצב והתנהגות
    - מצב = מבני נתונים - data members, attributes
    - התנהגות = פעולות / פונקציות - member functions, methods
- מחלקה (class)
  - מתאר (descriptor) של קבוצת עצמים, בעלי מאפיינים משותפים:
    - תכונות
    - פעולות
    - יחסים
    - התנהגות



מחלקות קיימות בקוד אך ורק **בזמן ההגדרה**
עצמים קיימים בזכרון אך ורק **בזמן הריצה**

מידול מונחה-עצמים של מרחב הבעיה

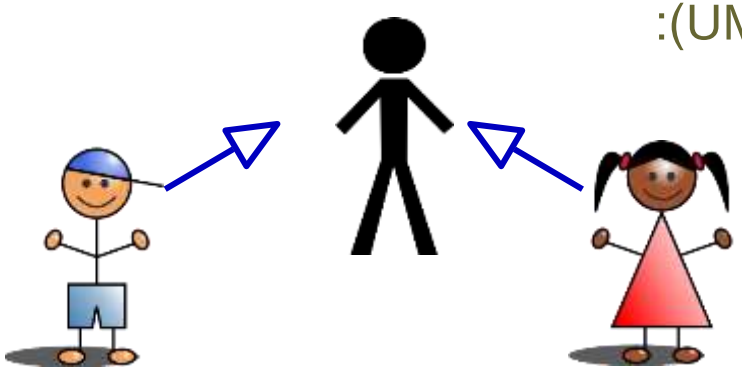
- הגישה מונחית-העצמים מאפשרת לבנות תוכנה מישויות המייצגות את מרחב הבעיה
  - בהמשך מוסיפים עליהן ישויות הנדרשות לפתרון
- כבר בשלב הניתוח המערכתי ניתן לבנות מודל מובנה של עצמים, המייצגים את מונחי מרחב הבעיה והקשרים ביניהם
  - PDOM = Problem Domain Object Model



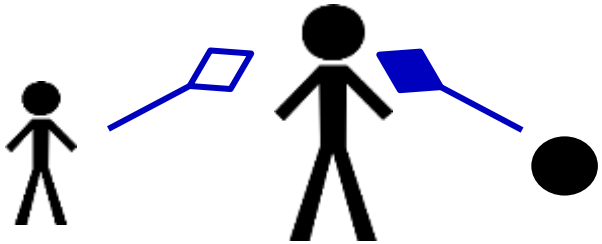
- המטרה
  - הבהרה וחידוד של המונחים והיחסים ביניהם
    - יצירת שפה משותפת בין בעלי העניין
- שימושים
  - יישוב סתירות ואי-בהירויות במפרטי הלקוח
  - מילון מונחים של המערכת
  - בסיס למודל מחלקות עבור התוכנה
    - ישויות המידע בהן נדרשת התוכנה לטפל
- הטכניקה
  - רשת סמנטית

רשת סמנטית

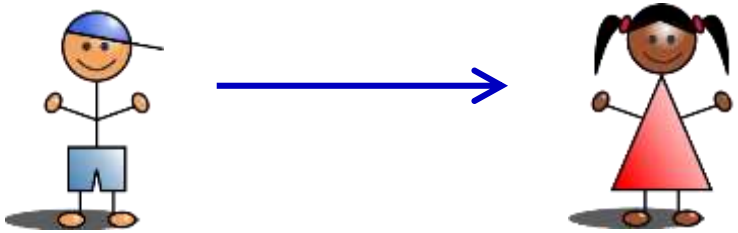
- רשת סמנטית היא אוסף של מושגים/ישויות והקשרים ביניהם
  - הקשרים הינם מהסוגים הבאים (סימוני UML):



- A is a B (או B is a_kind_of A)
  - a BOY is (a kind of) a PERSON
  - a GIRL is (a kind of) a PERSON



- A has a B (או B is_a_part_of A)
  - a HEAD is a part of a a PERSON
  - a PERSON has a CHILD



- B <relates to> A
  - a BOY loves a GIRL

מערכת המעליות – איתור ישויות במרחב הבעיה (מתוך האפיון התפעולי)

נוסע הנמצא **בקומה** כלשהי ורוצה להזמין **מעלית** לוחץ על **הכפתור המתאים לכיוון הנסיעה המבוקש**. אם לא היה דלוק קודם לכן, הכפתור נדלק בעקבות הלחיצה. מעלית כלשהי הנמצאת בכיוון הנסיעה המבוקש תגיע לקומה, תוך דקה לכל היותר. עם הגעתה נפתחת **הדלת** והכפתור כבה.

נוסע הנמצא בתוך המעלית ורוצה להגיע לקומה כלשהי לוחץ על **הכפתור המתאים לקומה**. אם לא היה דלוק קודם הכפתור נדלק בעקבות הלחיצה ולמעלית נוספת **בקשת עצירה**. הדלת נסגרת, לאחר השהיה, והמעלית ממשיכה בנסיעה, כאשר היא עוצרת בכל קומה עבורה קיימת בקשת עצירה. כאשר המעלית נעצרת בקומה הדלת נפתחת והכפתור המתאים לקומה כבה.

נוסע במעלית יכול לעצור את המעלית בזמן נסיעה באמצעות לחיצה על **כפתור עצירת החירום**. במקרה זה המעלית עוצרת מיד וכל בקשות העצירה שלה מתבטלות. לאחר מכן ניתן להחזיר את המעלית לפעולה על ידי לחיצה על כפתור עבור קומה כלשהי.

במקרה שהמעלית נתקעה במהלך נסיעה מזעיק הנוסע חילוץ באמצעות **כפתור החילוץ**. **המחלץ** (שהוא איש האחזקה של הבניין) מגיע **לפאנל החילוץ** שבחדר המכונות ומפעיל פקודות להורדת המעלית לקומת הקרקע ולפתיחת הדלת.

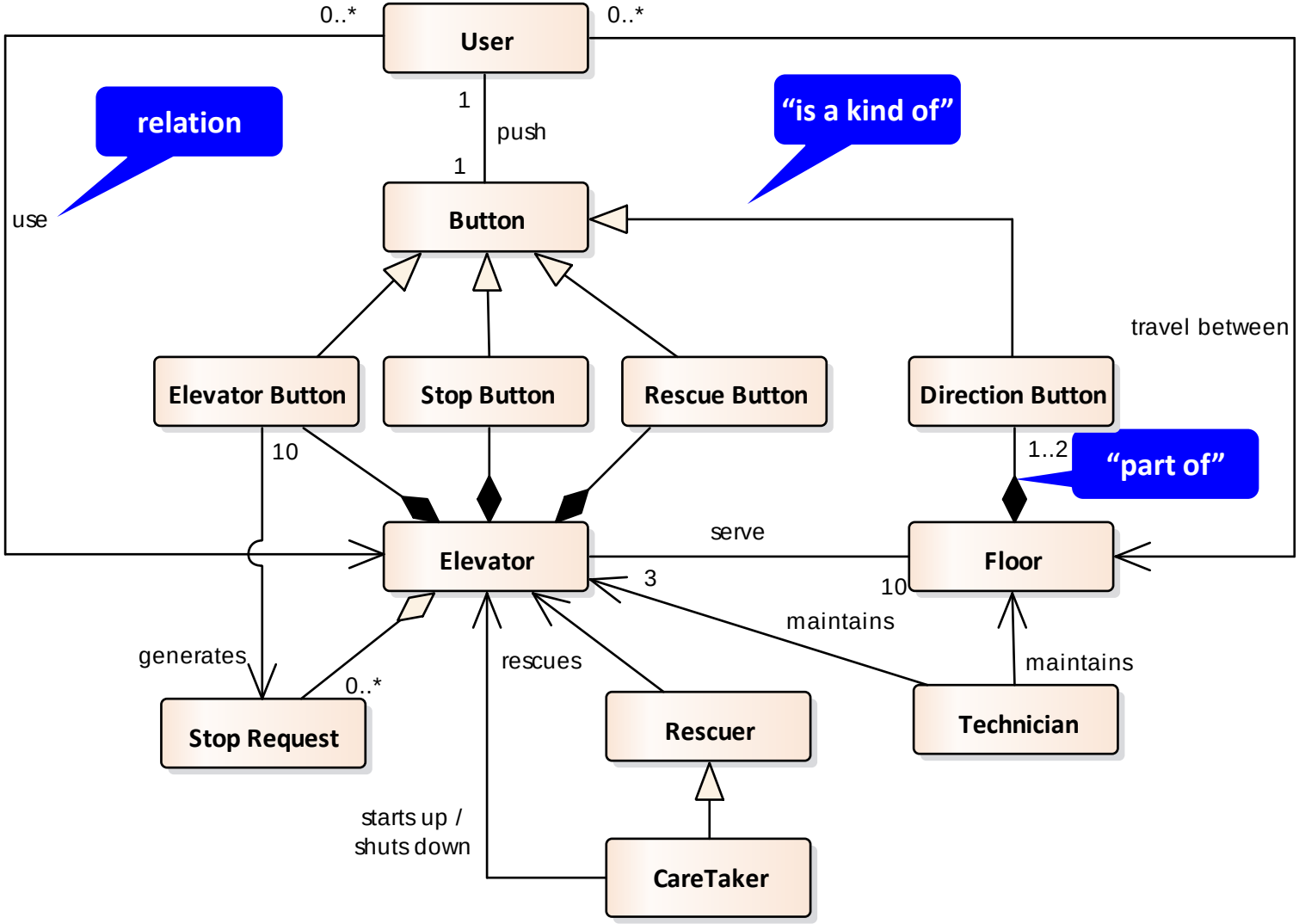
איש האחזקה אחראי להעלות את המערכת (Start Up) בתחילת יום פעילות ולהוריד אותה (Shut-Down) בסיומו.

טכנאי, המגיע אחת ל-6 חודשים, יכול לבצע **בדיקה** מקיפה של כל המערכת ולתקן **תקלות** באמצעות פיקוד **הטכנאי** שבחדר המכונות.

מערכת המעליות תעמוד בכל תקני הבטיחות הישימים.

המערכת תונגש לבעלי מוגבלויות שונות.

מערכת המעליות - PDOM



מטלת כיתה

- ערכו מודל מונחה-עצמים במרחב הבעיה (PDOM) למערכת ePark
  - השתמשו בארגז הכלים של class diagram
  - כללו בתרשים את הישויות הבאות:
    - Guardian
    - Account
    - Child
    - eTicket
    - Bracelet
    - Entry
    - Device
    - Supervisor

מרכיבי התוכנה: עצמים ומחלקות

- עצמים (objects)
  - היחידות הבסיסיות של התוכנה
  - כל עצם מנהל את המידע שבאחריותו באמצעות הפונקציונליות שהוקנתה לו
  - עצמים קיימים בזיכרון המחשב בזמן ריצת התוכנית
    - ניתן לבנות/להרוס עצמים באופן דינמי תוך כדי ריצה
      - constructor = פונקציה הבונה עצם חדש
      - destructor = פונקציה ההורסת (מוחקת) עצם קיים
    - לכל עצם יש מפתח גישה ייחודי (handle, pointer), הניתן לו ברגע בנייתו
- מחלקות (classes)
  - התבניות על פיהן נוצרים עצמים חדשים
  - התבנית מכילה 3 "תאים" (compartments)
  - המחלקות מוגדרות בקוד ע"י כותב התוכנה
- עצמים הם מופעים ספציפיים (instances) של מחלקות

שם
מאפיינים (מבני נתונים)
פעולות (פונקציונליות)

יצירה ותפעול של עצמים

Car

+ maker: string
+ model: string
+ licensePlate: string
+ testDate: Date
- owner: Person

+ sellTo(Person) : void
+ getOwner(int) : Person
+ testIsValid(Date) : boolean

יצירת עצם "מכונית" חדש

theBlueCar = new(Car)

איתחול פרטי המכונית

theBlueCar.maker = “mazda”

theBlueCar.model = “CX-7”

רישוי וטסט

theBlueCar.licensePlate = “12-345-67”

theBlueCar.testDate = 08/09/2010

מכירה

theBlueCar.sellTo(Lior)

function sellTo(X) { owner = X } ;

שם העצם

שם המחלקה

theBlueCar : Car
maker = “mazda”
model = “CX-7”
licensePlate = “12-345-67”
testDate = 08/09/2010
owner = Lior

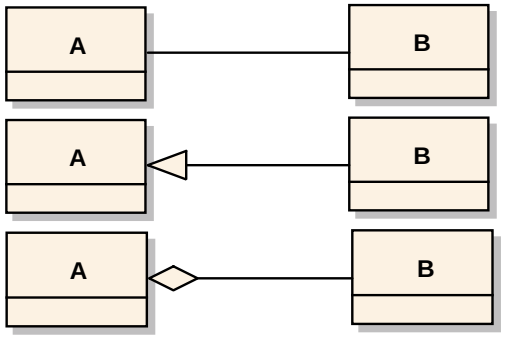
מועמדים לעצמים בתוכנה

- עצמים המייצגים ישויות פיזיות (מנוע, דלת, עמדת עבודה, ...)
  - מאפיינים: פרמטרים ונתונים לגבי הישות, קלט/פלט
  - מתודות: פונקציונלית פיזית
    - העצם המייצג משמש, למעשה, כממשק שבין התוכנה לישות הפיזית
- עצמים המייצגים ישויות לוגיות (תהליך, שירות, ...)
  - מאפיינים: פרמטרים ונתונים לגבי הישות, קלט/פלט
  - מתודות: פעולות המשמשות את התהליך/השירות
- עצמים המייצגים ישויות מידע (מאגרי נתונים, רשימות, תורים, ...)
  - מאפיינים: רכיבי המידע שבאחריות הישות
  - מתודות: פעולות על המידע (אחסון, שליפה, עדכון, ...)

תרשים מחלקות (Class Diagram) - תחביר

ClassName
- privateAttribute: Type + puplicAttribute: Type
- privateMethod(X:TypeX, Y:TypeY) : ReturnType + publicMethod(X:TypeX, Y:TypeY) : ReturnType

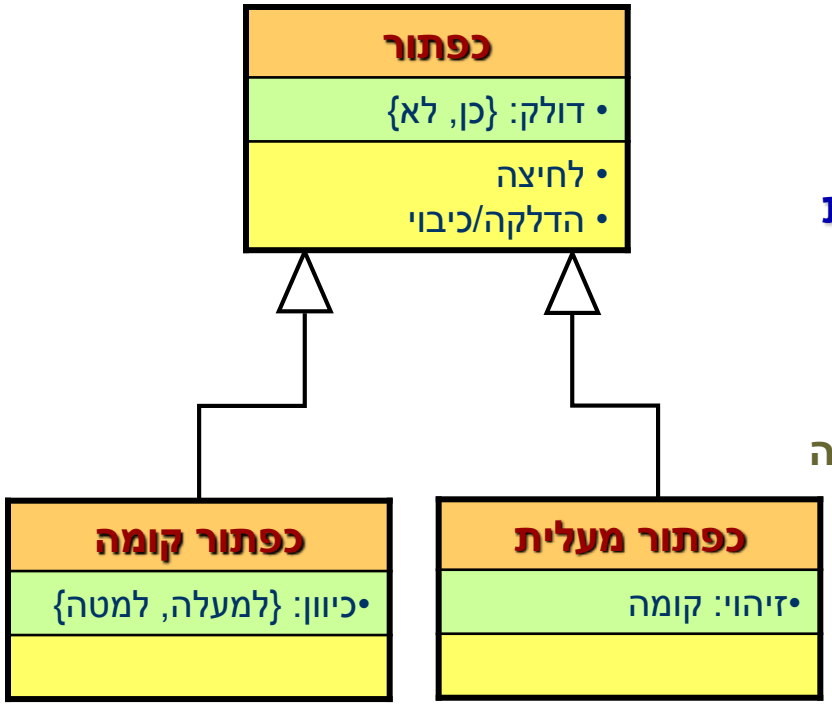
- מחלקה
  - שם המחלקה
  - מאפיינים (משתנים)
    - מאפיין פרטי (-): ניתן לגשת אליו רק מתוך המחלקה עצמה
    - מאפיין ציבורי (+): ניתן לגשת אליו גם מבחוץ
  - מתודות (פונקציות)
    - מתודה פרטית (-): ניתן לקרוא לה אך ורק מתוך המחלקה עצמה
    - מתודה ציבורית (+): ניתן לקרוא לה גם מבחוץ
- קשרים
  - זיקה (association)
  - ירושה (Inheritance)
  - הקבצה (aggregation)



- תרשים המחלקות מבוסס על עיקרון של רשת סמנטית

יחס ירושה (inheritance) / הכללה (Generalization)

- כאשר מחלקה B יורשת/מכלילה את מחלקה A, אזי
  - B מכילה את כל המאפיינים של A
  - B מכילה את כל הפעולות של A
  - בנוסף, B מכילה מאפיינים ופעולות משל עצמה
- ירושה מתארת את היחס הסמנטי "B is-a A"
- B היא תת-מחלקה (sub-class) של A
  - מינוח לא מוצלח, כי B מכילה יותר מאשר A
- **יחס הירושה יוצר מבנה היררכי של מחלקות**
- **מחלקה אבסטרקטית**
  - מחלקה שלא ניתן ליצור ממנה עצמים
  - **כל העצמים נוצרים ממחלקות היורשות אותה**
  - לדוגמה: "כלי רכב"

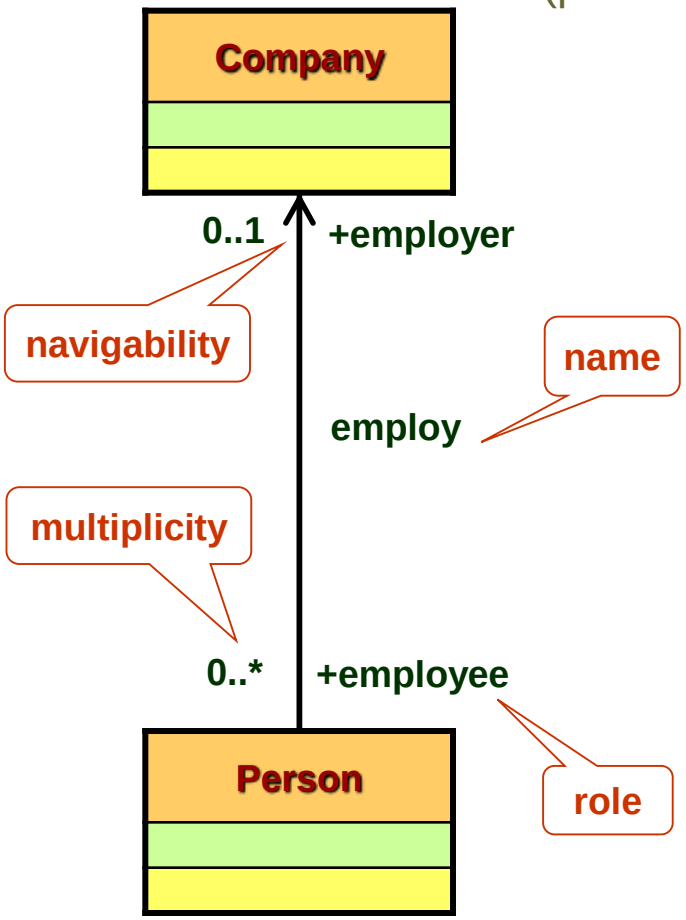


בעיות ביחסי ירושה

- ירושה מרובה (multiple inheritance)
  - מחלקה אחת יורש משתי מחלקות שונות
  - הבעיה: עלולות להיווצר סתירות במאפיינים/פעולות
  - הפתרון: רוב שפות התכנות אינן מרשות ירושה מרובה (מבנה של עץ)
- ירושה עמוקה מדי
  - A ⟶ B ⟶ C ⟶ ... ⟶ X
  - הבעיה: קושי במעקב אחר הקשר (קשיי תחזוקה)
  - הפתרון: "שבירת" העץ בנקודות בהן הזיקה חלשה יותר
- ירושה מדומה
  - לדוגמה: ריבוע הוא סוג של מלבן, לכן מלבן ⟶ ריבוע
  - הבעיה: למלבן יש שני מאפיינים (אורך, רוחב) ולריבוע רק אחד (צלע)
  - הפתרון: להגדיר את הירושה על בסיס תכונות משותפות (מאפיינים/פעולות)

זיקה - association

- יחס בין **מחלקות** המגדיר "היכרות" בין **עצמים** ממחלקות אלה (Company,Person)
  - ה"היכרות" הינה באמצעות מצביעים (pointers,references)
- מאפיינים המאפשרים הגדרה ברורה יותר של זיקה
  - שם היחס (name)
    - עשוי להתפרש בכיוונים שונים
      - "Company employs Person"
      - "Person employed by Company"
  - תפקיד (role)
    - "Company" is the employer
    - "Person" is the employee
  - ריבוי (multiplicity)
    - "Company employs 0 or more Persons"
    - "Person is employed by 0 or 1 company"
  - ניווט (navigability)
    - Person "knows" who is its Company
    - Company does not know its Persons



הקבצה (aggregation)

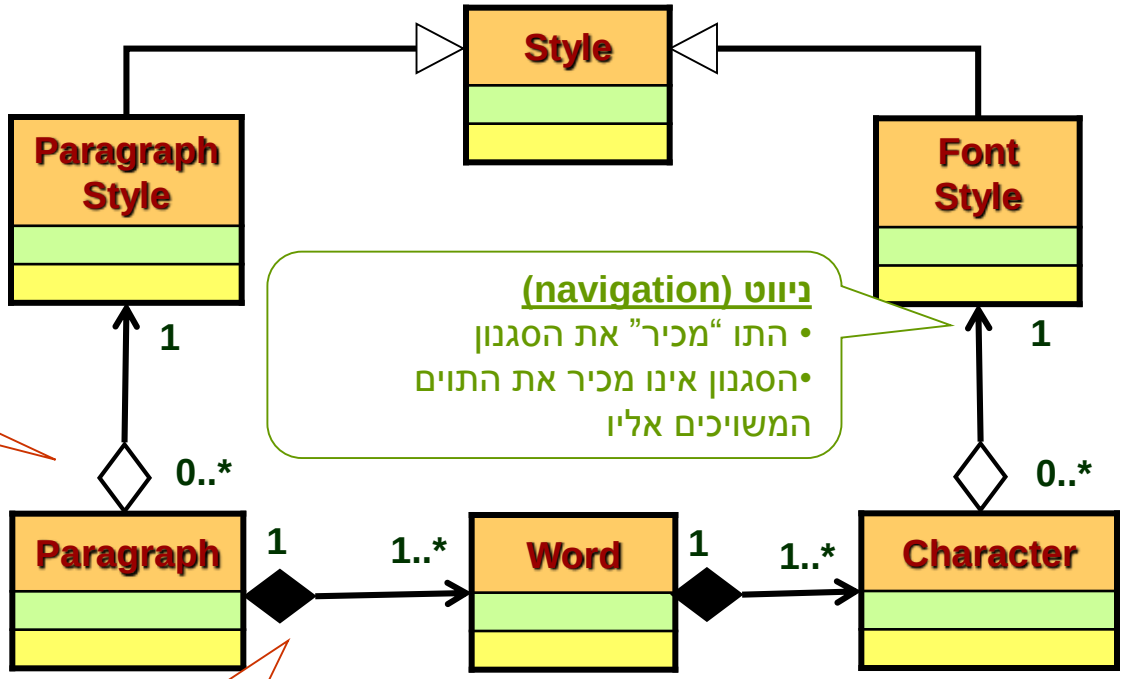
- סוג מיוחד של זיקה (היכרות בין עצמים)
  - מתארת את היחס הסמנטי "B has_a A)
- שני סוגי הקבצה:
  - הקבצת הרכֵּב (composite aggregation)

    - A הוא חלק **בלתי נפרד** מ-B, **ורק מ-**B
      - קיומו של A תלוי בקיומו של B
    - שמות נוספים:
      - whole-part aggregation
      - non-shared aggregation
  - הקבצת שיתוף (shared aggregation)

    - A משוייך ל-B, אבל
      - קיומו של A אינו תלוי בקיומו של B
      - A יכול להיות משותף, כלומר משוייך בו זמנית גם לעצמים אחרים

הקבצה - דוגמה

הקבצת שיתוף (shared)
- לפיסקה יש סגנון-פיסקה אחד
- סגנון יכול להיות משותף למספר פיסקאות
- הסגנון הוא ישות עצמאית, וקיומו אינו מותנה בקיום פיסקאות
- מחיקת פיסקה אינה מוחקת את הסגנון



ניווט (navigation)
- התו "מכיר" את הסגנון
- הסגנון אינו מכיר את התוים המשויכים אליו

הקבצת הרכב (composite)
- פיסקה מורכבת ממילה אחת לפחות
- כל המילים של הפיסקה שייכות אך ורק לפיסקה זו
- קיומה של המילה מותנה בקיומה של הפיסקה
- מחיקת פסקה מוחקת את כל המילים המרכיבות אותה.

1. A Numbered Title

This is the first *paragraph* of this document. It contains 17 *words* and 80 non-blank characters.

אנליזה פונקציונאלית ברמת התוכנה

- בדומה לתהליך האנליזה הפונקציונאלית שעשינו בשלבים הקודמים, גם כאן עלינו להקצות פונקציונאליות למרכיבי התוכנה (עצמים/מחלקות)
  - בשלב ראשון מרכיבי התוכנה הם המחלקות שזוהו ב-PDOM
    - כעת יש להקצות להם פונקציונאליות (מאפיינים ומתודות)
  - בהמשך התהליך ניתן לזהות ולהוסיף מחלקות בהתאם לצורך
- מקורות לפונקציונאליות
  - ארכיטקטורת התוכנה – מימוש התהליכים באמצעות sequence diagrams
- **בהקצאה הפונקציונאלית יש לשמור על עקרונות המקצוענות והעצמאות**
  - לכידות הדוקה: מה המשותף בין כל המאפיינים וכל המתודות שהוקצו למחלקה?
  - צימוד רופף: עד כמה המחלקה תלויה במחלקות אחרות?

אנליזה פונקציונאלית ברמת המערכת

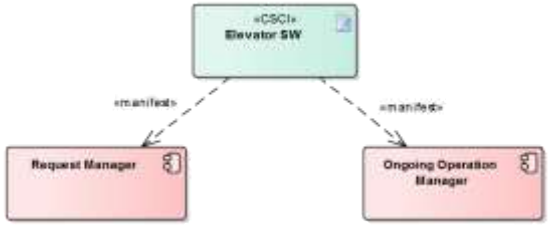
תזכורת

- בשלב הארכיטקטורה המערכתית עשינו את הצעדים הבאים:

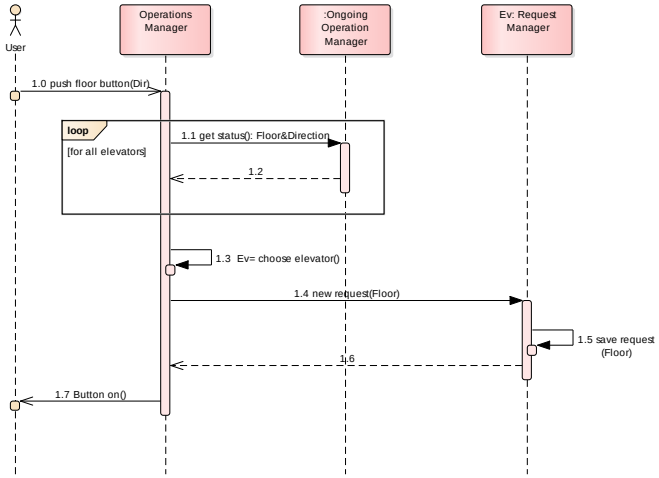
1. מיצוי פונקציונאליות מתוך תרחישים

SUC-1	הזמנת מעלית
שחקנים ויעדים	נוסע: לקבל מעלית זמינה לנסיעה
ב"ע ואינטרסים	אין
Pre-conditions	• הנוסע נמצא בקומה כלשהי בה נמצאת דלת של מעלית • המערכת פעילה [Post-cond. של UC "איתחול מערכת"]
Post-Conditions	• מעלית פתוחה נמצאת בקומה בה נמצא הנוסע (יעד)
Trigger	• הנוסע לוחץ על כפתור עליה / ירידה בקומה
MSS	1. המערכת קולטת את הלחיצה 2. הכפתור נדלק 3. המערכת מאתרת מעלית הנוסעת בכיוון המבוקש 4. המערכת מקצה את העצירה למעלית 5. המעלית מגיעה לקומה 6. דלת המעלית נפתחת 7. כפתור הקומה כבה
הסתעפות א'	חלופה בצעד 2 של MSS: הכפתור כבר דלוק (כבר הוקצתה מעלית) 2א1. מעבר לצעד 5
עקיבות לדרישות	...

2. פירוק לרכיבים (<<manifest>>) והקצאת פונקציות



פונקציות הקשורות בניהול הנסיעה השוטפת של מעלית	פונקציות הקשורות באיסוף ועדכון בקשות העצירה של המעלית
• ניהול מהלך הנסיעה • הסעת/עצירת מנוע • פתיחת/סגירת דלת • עצירת חירום • אזעקת חילוץ	• הגבה ללחיצות על כפתורים במעלית • הדלקת/כיבוי כפתורים במעלית • רישום בקשות מכפתורים • רישום בקשות מטכנאי • רישום הקצאות נסיעה מהפיקוד המרכזי



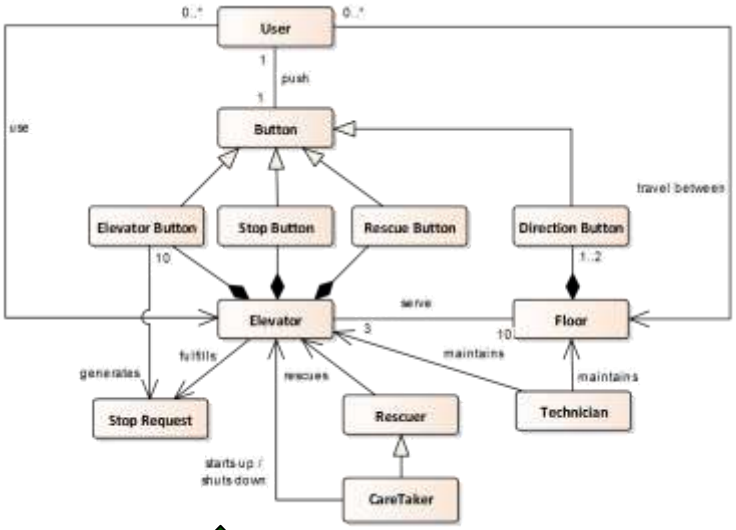
3. מימוש התרחישים כאינטראקציה בין רכיבים (Sequence Diagrams)

אנליזה פונקציונאלית ברמת התוכנה

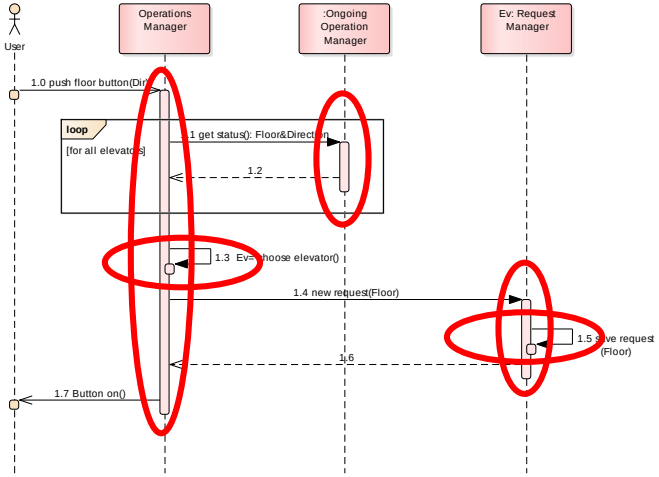
- בשלב הנוכחי (תכן תוכנה) ניתן לחזור על התהליך ברמה אחת פנימה

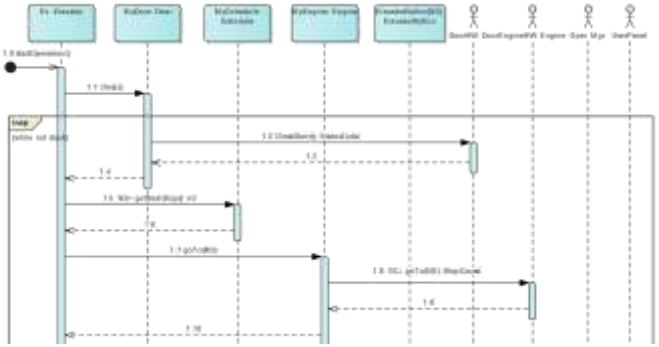
1. מיצוי פונקציונאליות מתוך Seq. Diag.

2. זיהוי עצמים (Class Diag.)
והקצאת פונקציונאליות (attr., methods)



מפורט בשקף הבא

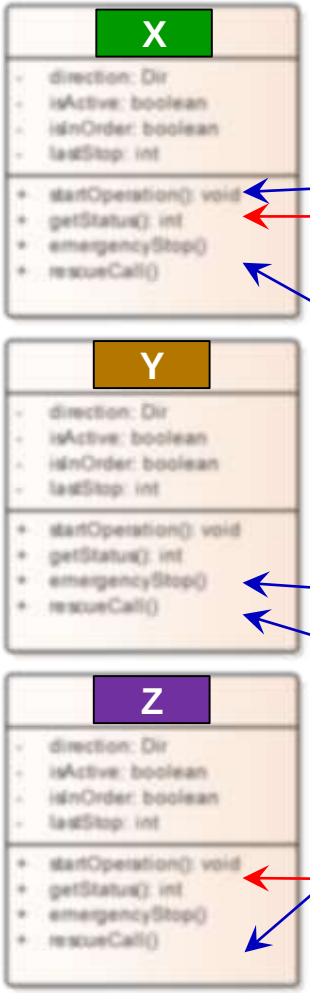




3. מימוש התרחישים
כאינטראקציה בין עצמים

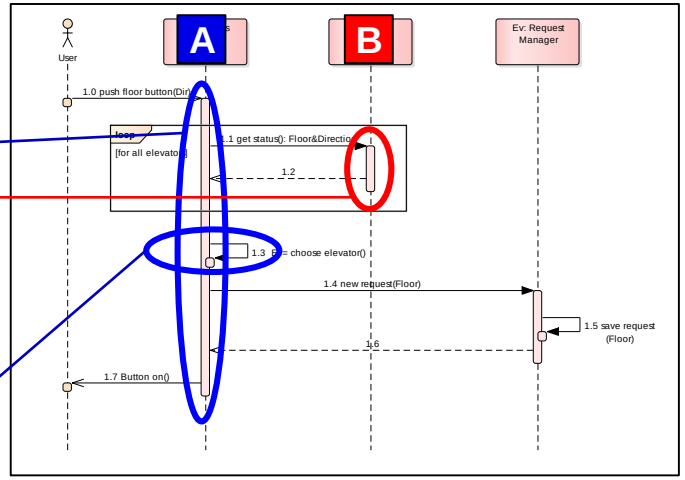
הקצאת פונקציונאליות הרכיבים למחלקות התוכנה

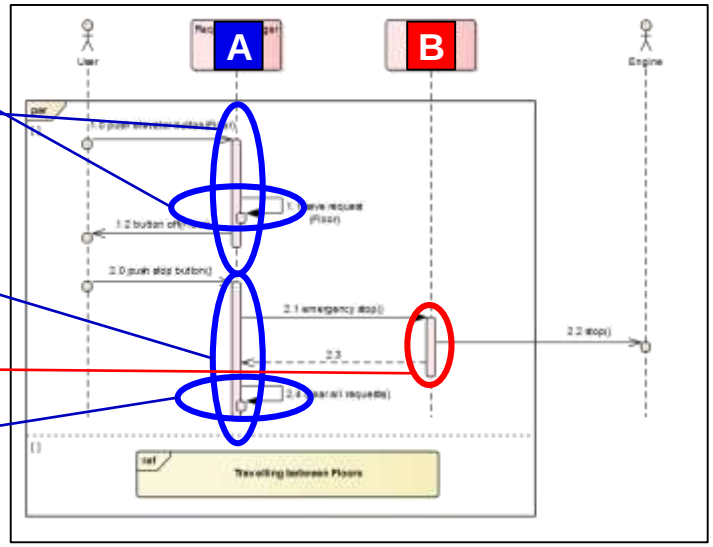
מחלקות שזוהו עד עתה
(בשלב ראשון – PDOM)



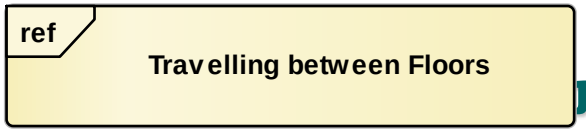
Build A = X+Y+Z+NewClass
Build B = X+Z

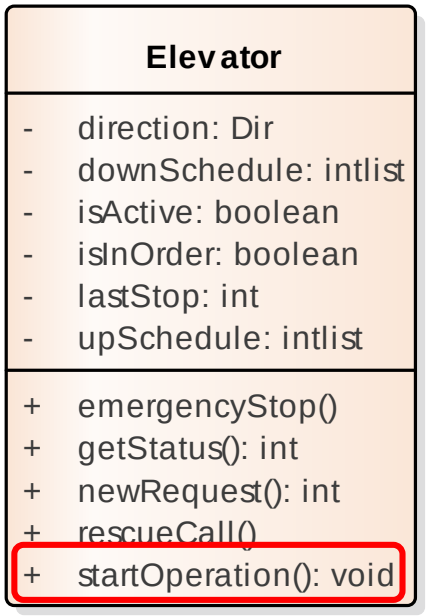
SD בהם פעילים רכיבים A ו-B

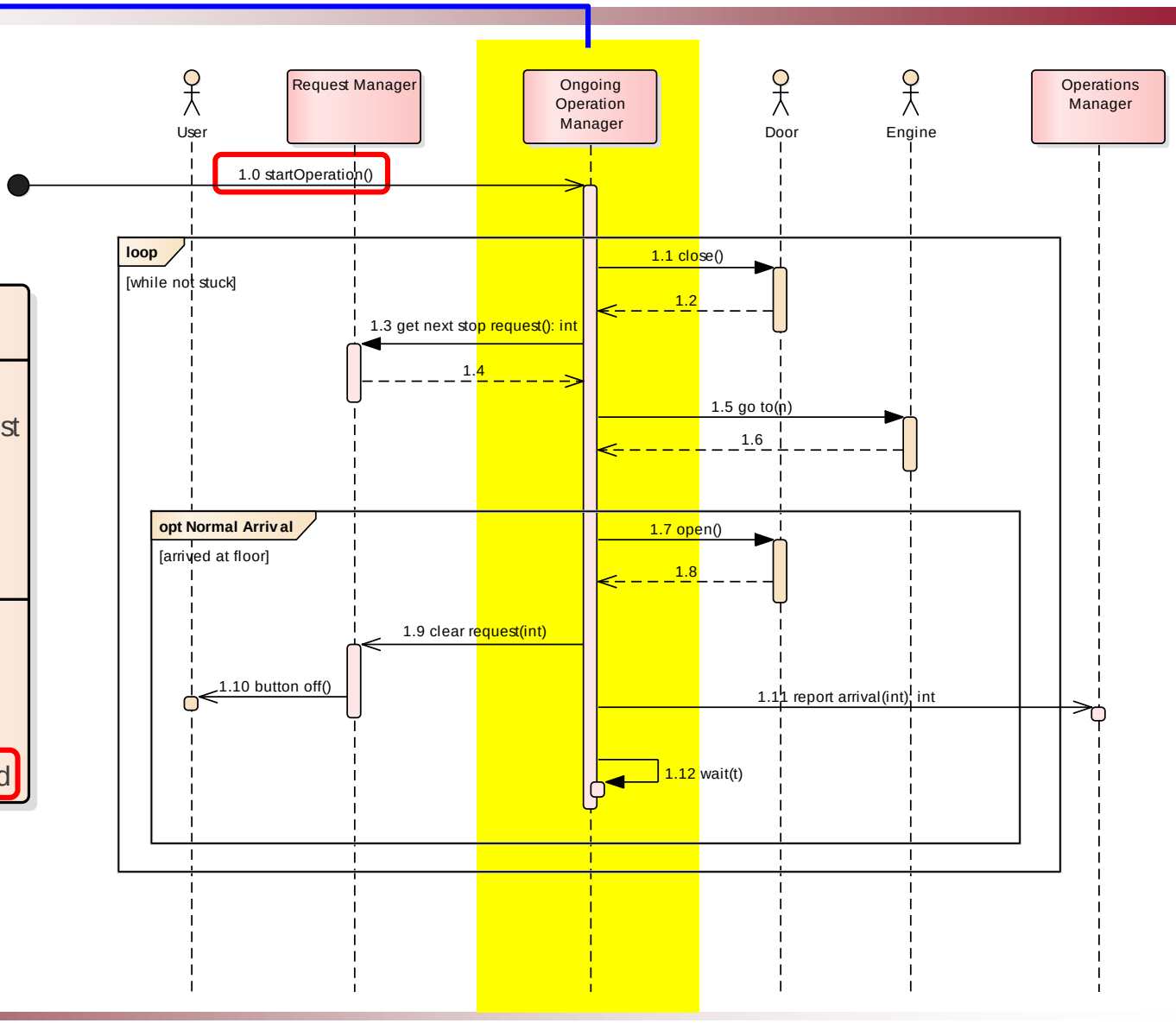




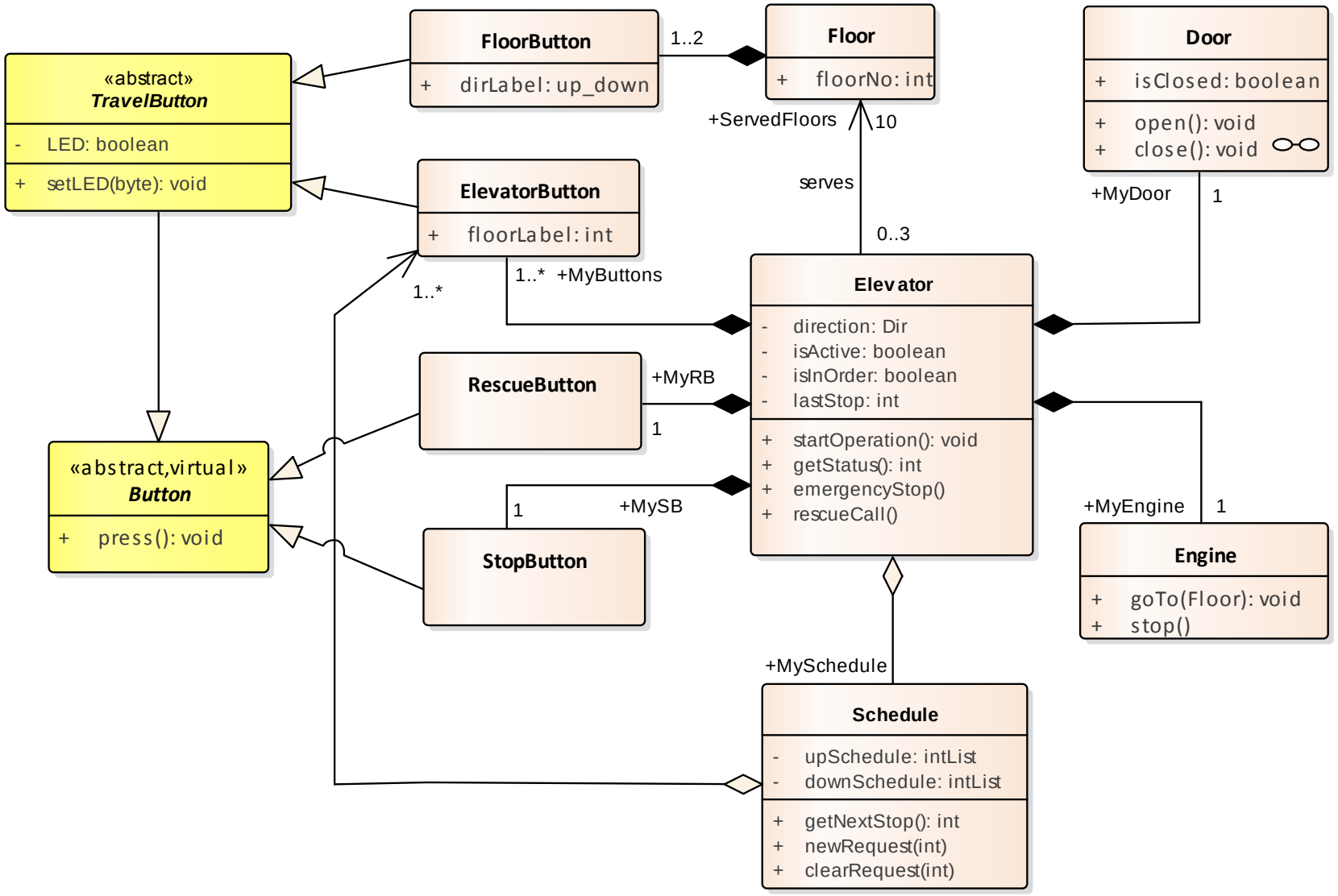
דוגמה: תרחיש הפעולה השוטפת של המעלית







מערכת המעליות - תרשים מחלקות תשתיתיות, על בסיס ה-PDOM



מערכת המעליות – מחלקת "מעלית"

Elevator

- direction: Dir
- downSchedule: intlist
- isActive: boolean
- isInOrder: boolean
- lastStop: int
- upSchedule: intlist

+ emergencyStop()
+ getStatus(): int
+ newRequest(): int
+ rescueCall()
+ startOperation(): void

בנוסף למאפיינים אלה יש גם מצביעים למחלקות אחרות, הנגזרים מיחסי זיקה/הקבצה

בנוסף למאפיינים ולמתודות אלה יש גם מאפיינים ומתודות דרך ירושה

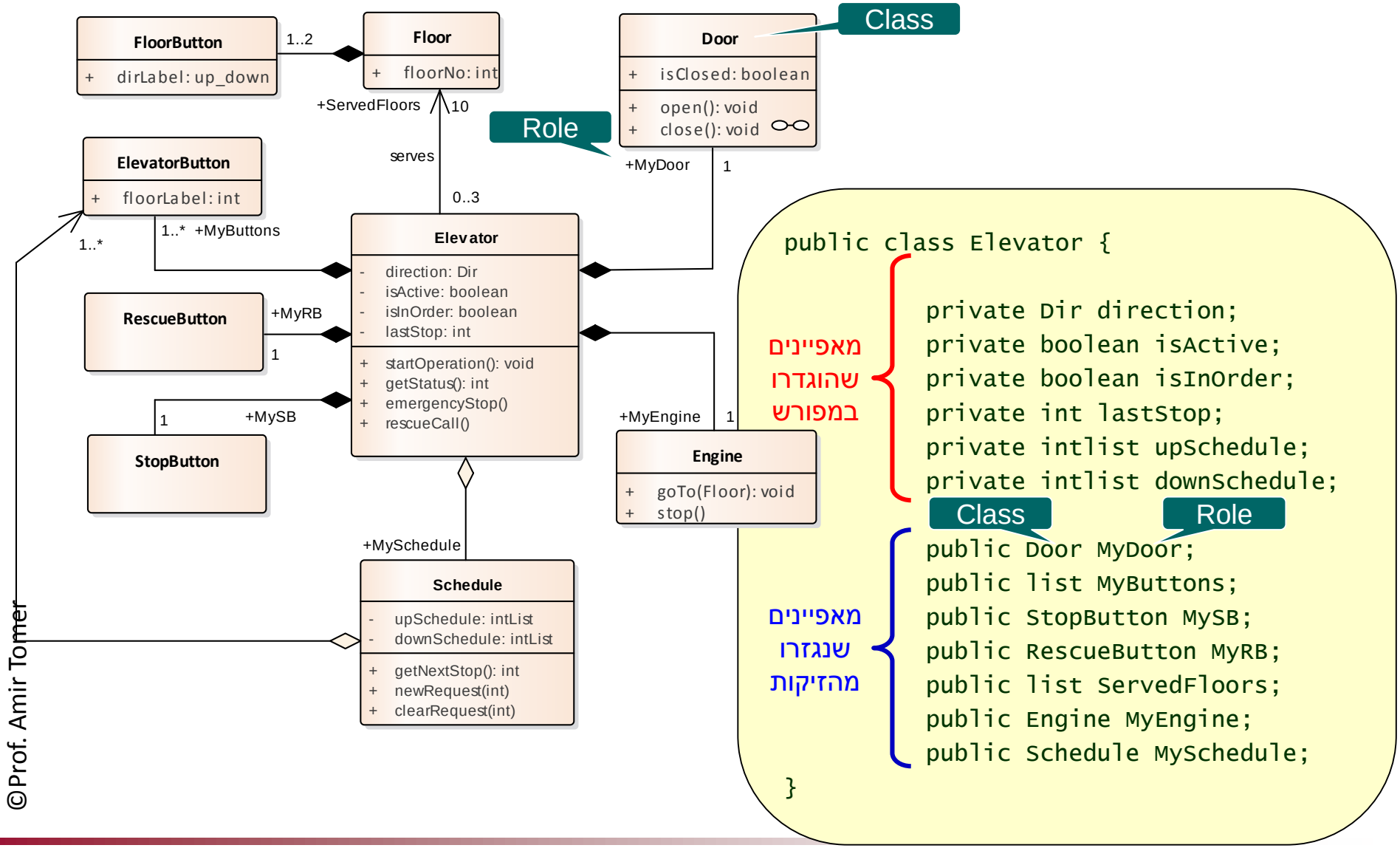
מטלת כיתה

- ערכו מודל מחלקות תשתית עבור ePark
  - השתמשו בישויות שהגדרתם ב-PDOM כמחלקות
  - הוסיפו למחלקות מאפיינים (attributes) על בסיס שיקול דעתכם
  - הוסיפו בכל מחלקה getter ו-setter לאחד מהמאפיינים כמתודות (operations)
  - אם כבר בשלב זה ידוע לכם על מתודות שיידרשו למחלקה – הוסיפו גם אותן

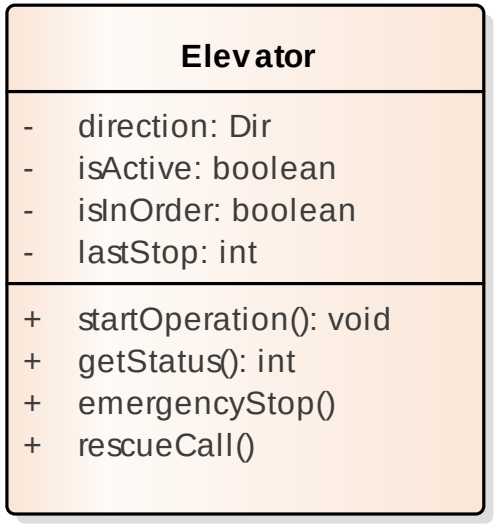
עקיבות הדרישות הפונקציונליות למודל המחלקות

- המחלקות שבמודל המחלקות אמורות לספק את כל הפונקציונליות המערכתית
- מכל דרישה פונקציונלית בטבלת הדרישות יש להצביע למחלקה או למחלקות הרלוונטיות
  - משתתפות בדרישה תפעולית (OR)
    - לדוגמה: "אם לא היה דלוק קודם נדלק הכפתור בעקבות הלחיצה" ← כפתור
  - מספקות את מבני הנתונים עבור דרישות המידע (DR)
    - לדוגמה: "בכל קומה יהיו שני כפתורים" ← קומה
- מכל מחלקה במודל המחלקות יש להצביע על הדרישות הפונקציונליות הרלוונטיות לה

הפקה אוטומטית של קוד סטטי ממודל המחלקות - Attributes



הפקה אוטומטית של קוד סטטי ממודל המחלקות - Methods



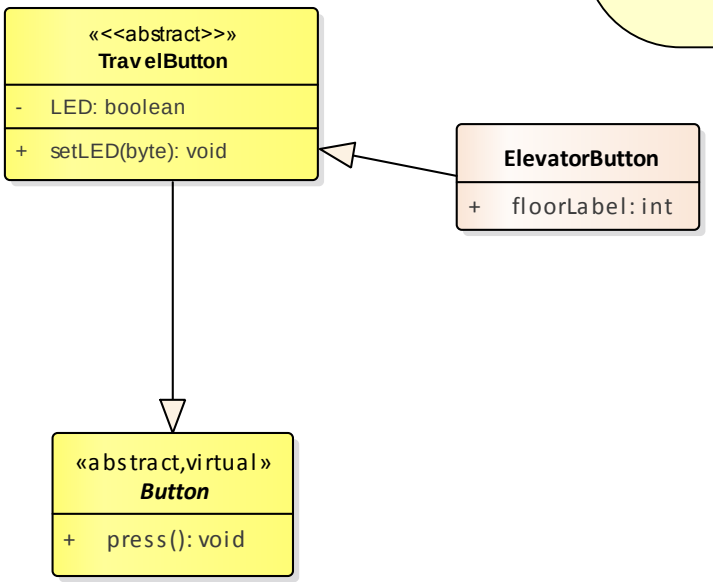

```
public class Elevator {

    public Elevator(){ }
    public void finalize() throws Throwable { }
    public void startOperation(){ }
    public int getStatus(){
            return 0;
    }
    public emergencyStop(){}
    public rescueCall(){}
```

Constructor

Destructor

מתודות שהוגדרו במפורש

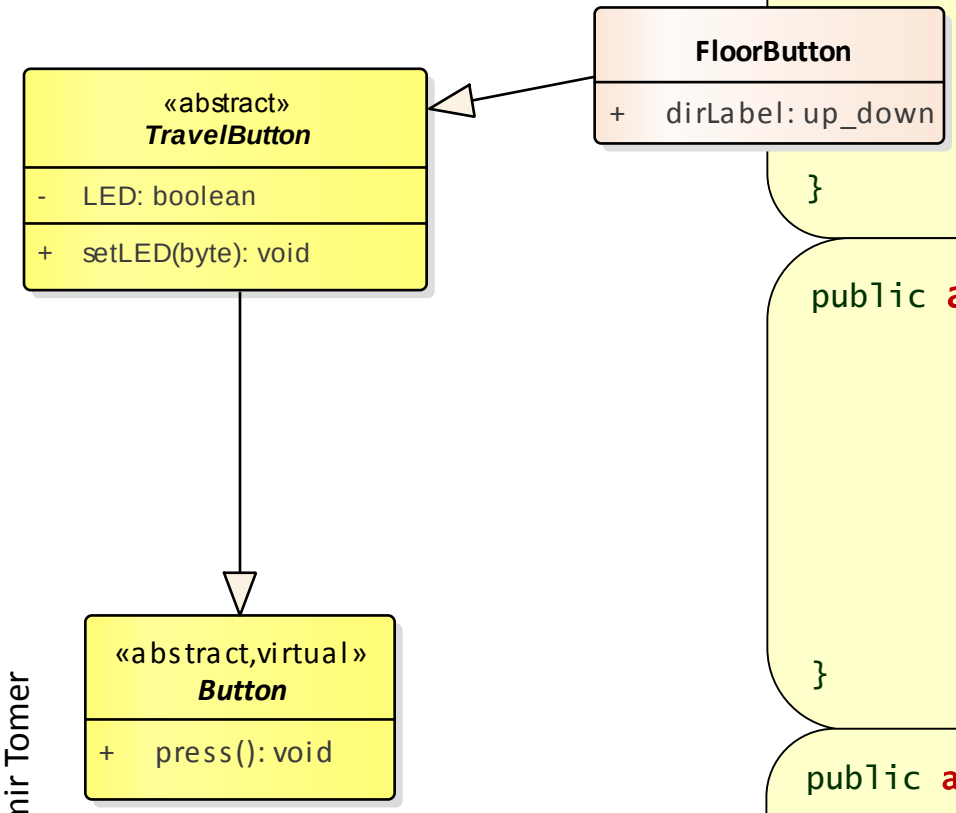



```
public class TravelButton extends Button {

        private boolean LED;
        public TravelButton(){ }
        public void finalize() throws Throwable
                super.finalize();
        }
        public void setLED(byte on_off){ }

}
```

הפקה אוטומטית של קוד סטטי ממודל המחלקות – ירושות




```
public class FloorButton extends TravelButton {
        public up_down dirLabel;
        public FloorButton(){ }
        public void finalize() throws Throwable {
                super.finalize();
        }
}
```

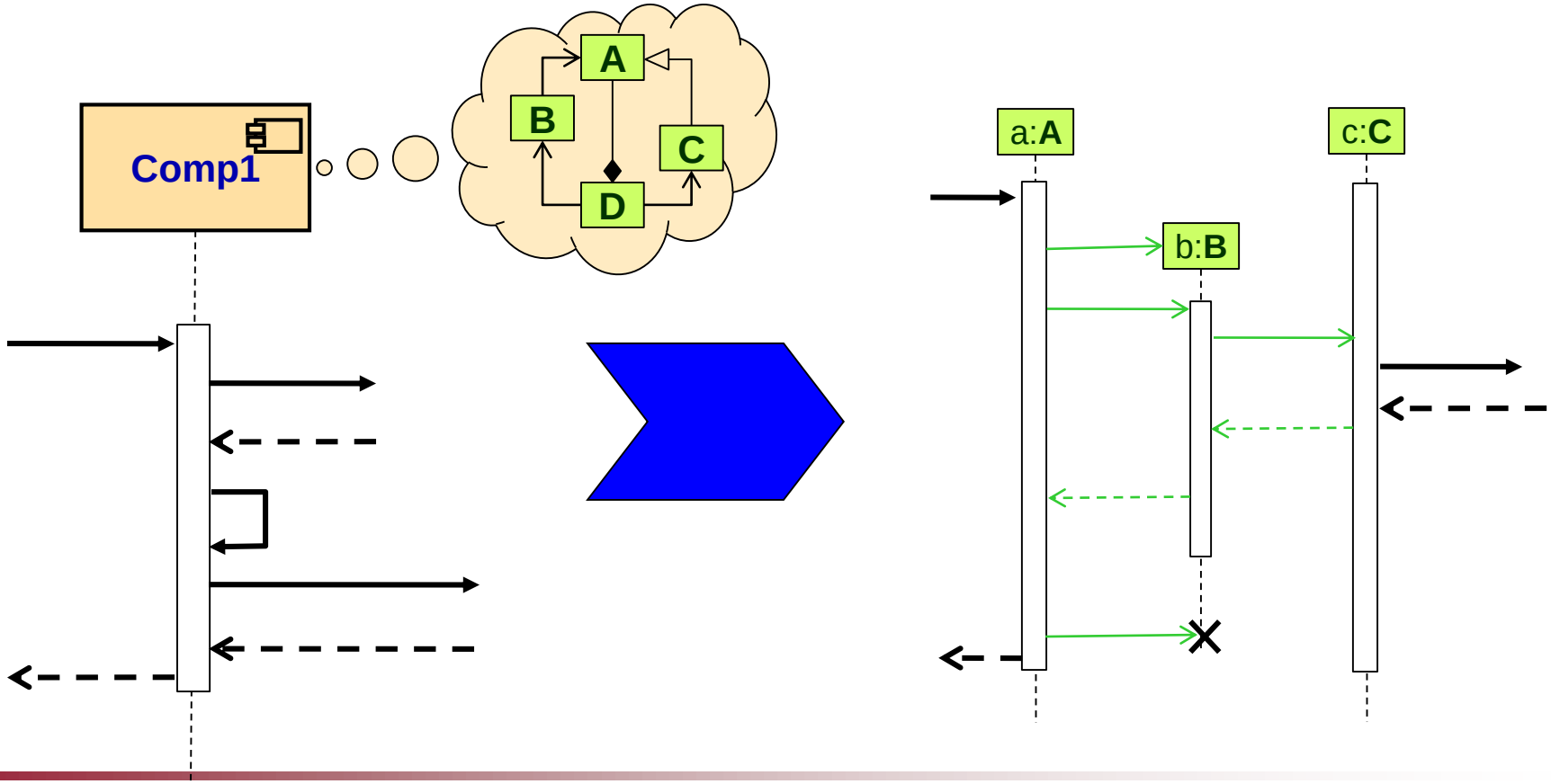
```
public abstract class TravelButton extends Button {

        private boolean LED;
        public TravelButton(){ }
        public void finalize() throws Throwable {
                super.finalize();
        }
        public void setLED(byte on_off){ }
}
```

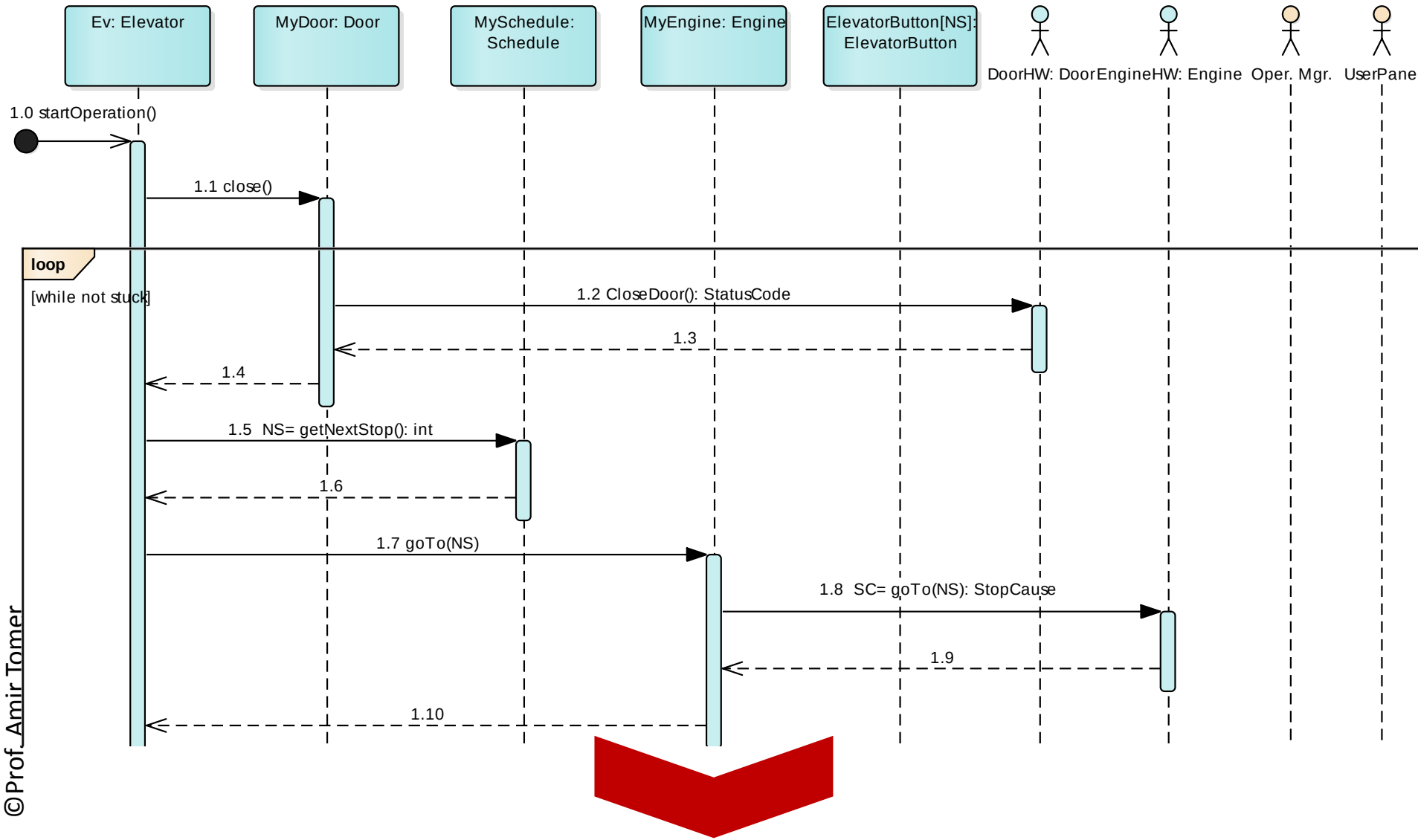
```
public abstract class Button {
        public Button(){ }
        public void finalize() throws Throwable { }
        public void press(){ }
}
```

מימוש תהליכי התוכנה

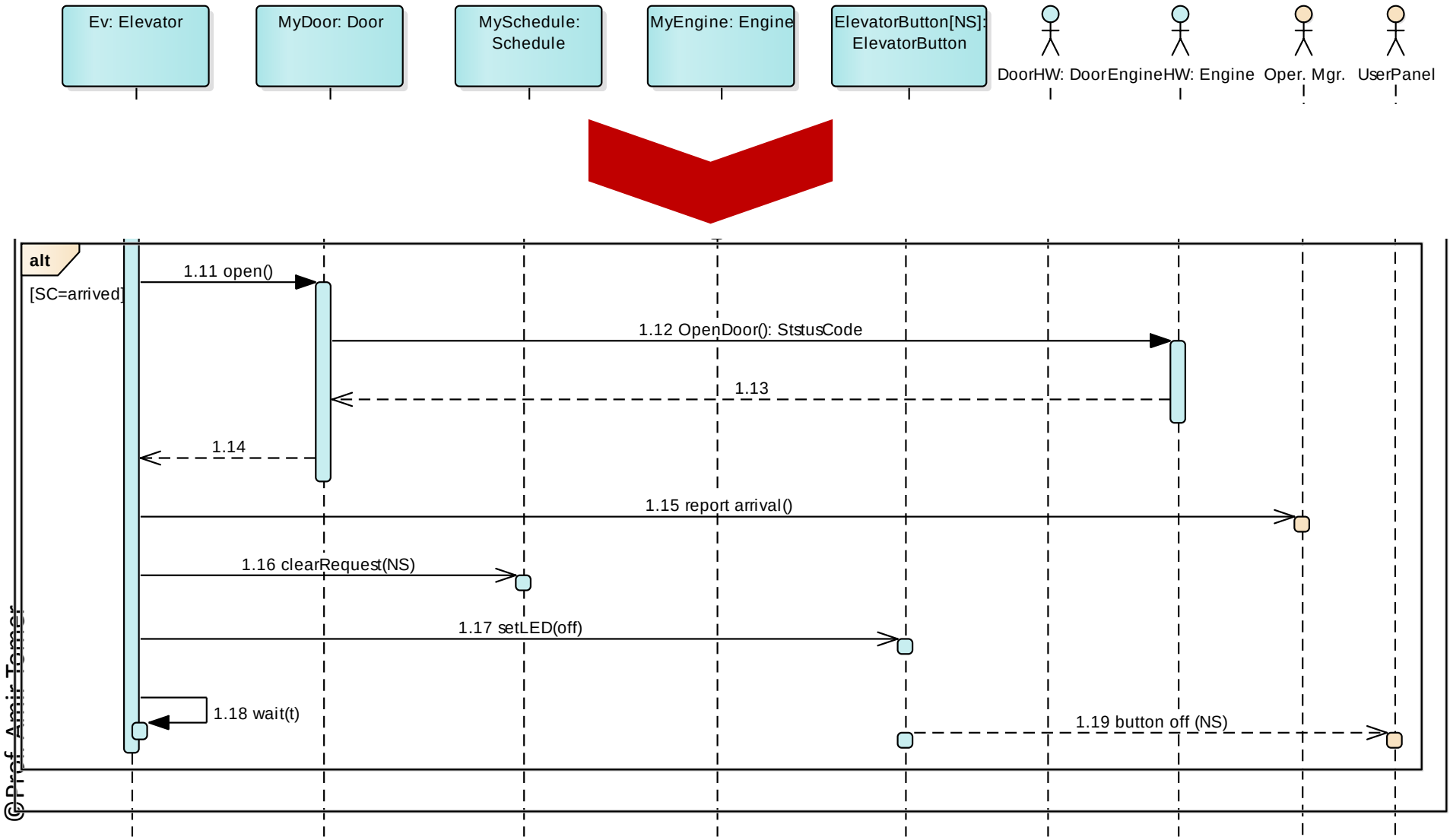
- מימוש הפונקציונאליות של כל רכיב באמצעות עצמים בתוכנה
  - פונקציונאליות הרכיב ← תפקודו במסגרת מימוש התהליכים המערכתיים
  - עצמים בתוכנה ← על בסיס מודל המחלקות



מימוש פעילות OngoingOperation של רכיב Ongoing Operation Manager באמצעות העצמים המרכיבים אותו



מימוש פעילות OngoingOperation של רכיב Ongoing Operation Manager באמצעות העצמים המרכיבים אותו (המשך)



מטלה: Sequence Diagram ברמת עצמים

- SUC-1 של ePark מתאר את תהליך ההרשמה בכניסה לפארק. במסמך SAD מופיע sequence diagram ברמת רכיבים של תהליך זה
- ערכו sequence diagram ברמת עצמים של התהליך (כלומר, מהלך הפעילות ()new registration של רכיב Usage Manager) באמצעות עצמים
  - תוך כדי בניית התרשים הוסיפו את המתודות הנדרשות למחלקות של העצמים המשתתפים
  - פניות לרכיבים אחרים ימומשו כמתודות עצמיות

מודל מחלקות לרכיב הבודד

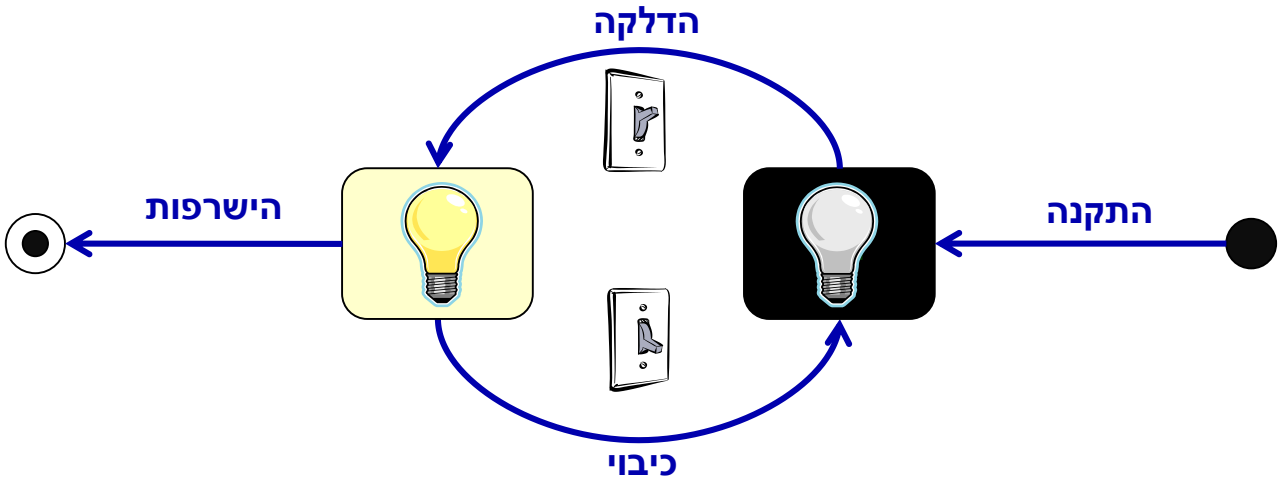
- ניתן לבנות מודל מחלקות ספציפי לכל רכיב
- המודל **עשוי** לכלול מחלקות מהסוגים הבאים:
  - תת-קבוצה של מחלקות התשתית
    - אלה שהרכיב צריך לצורך פעולתו
      - למשל: מחלקת "כפתור מעלית" עבור רכיב Ongoing Operation Manager
  - מחלקות היורשות מחלקות תשתית
    - עידונים ספציפיים של מחלקות התשתית לצורך פעולתו של הרכיב
      - למשל מחלקת "לחצן גרפי" היורש "כפתור מעלית"
  - מחלקות המייצגות ישויות מחוץ לרכיב
    - מחלקות Proxy לצורך קשר עם העולם הפיזי
      - למשל מחלקת UserPanelProxy הקולטת את לחיצות המשתמש (פסיקות למערכת) ומפנה אותן לעצמים המייצגים את הכפתורים המתאימים
    - מחלקות Proxy המייצגות רכיבים אחרים
      - למשל מחלקת RequestManagerProxy המייצגת את רכיב Request Manager עבור הרכיב Ongoing Operation Manager

דלת המעלית – דרישות משלימות / נגזרות

- במעלית יש שני כפתורים עבור הדלת
  - Open Door (OD)
  - Close Door (CD)
- בדלת יש עינית (detector) המזהה מעבר של גוף דרך הדלת
- בעת פתיחה או סגירה של דלת תושמע הודעה קולית מתאימה
- כאשר המעלית לא פעילה היא חונה בקומה כלשהיא עם דלת סגורה
- לחיצה על כפתור קומה בה חונה מעלית תגרום לפתיחת הדלת
- דלת פתוחה תיסגר אחרי השהיה נתונה או כאשר נלחץ כפתור CD
- המעלית יכולה לנסוע רק כאשר הדלת סגורה במלואה
- כל עוד המעלית בתנועה שום אירוע לא יגרום לפתיחת הדלת
- לאחר עצירת מעלית בקומה כלשהיא הדלת נפתחת
- כל עוד הדלת לא סגורה כל אחד מהאירועים הבאים יביא לפתיחתה:
  - לחיצה על כפתור OD
  - זיהוי של העינית
  - לחיצה על כפתור קומה בקומה בה היא נמצאת

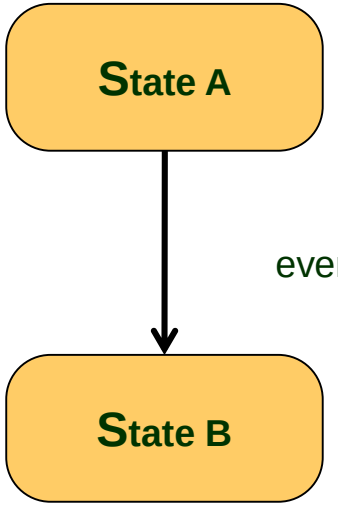
מודל מכונת-מצבים (State-Machine Model)

- מכונת מצבים
  - "אוטומט" (מודל מתימטי)
  - מודל דינמי לתיאור התנהגות
- שימושי המודל
  - ברמה המערכתית – אפיון ההתנהגות הכוללת של מערכת
  - ברמת המחלקה – מחזור החיים של אובייקט

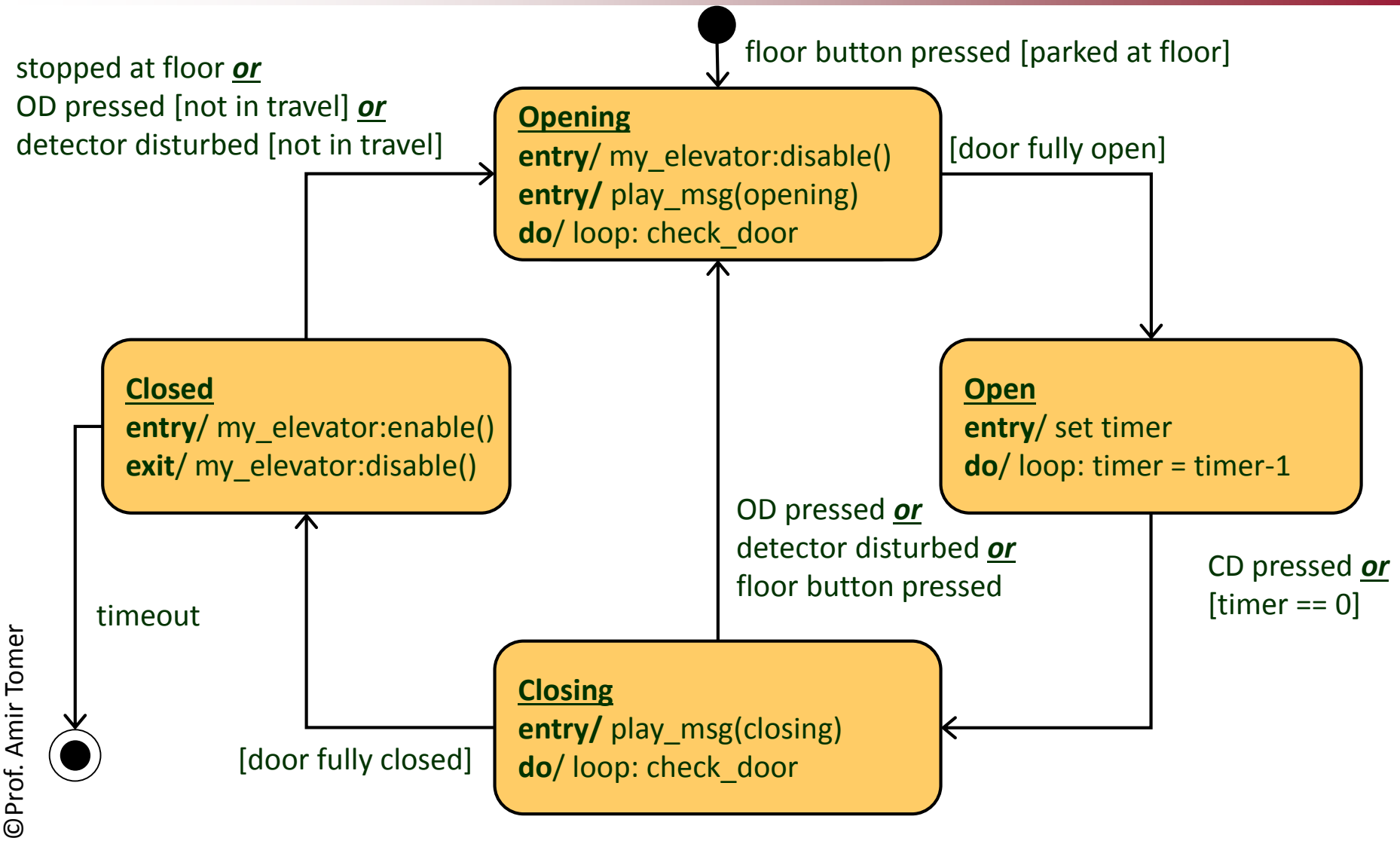


תרשים מצבים – State Chart

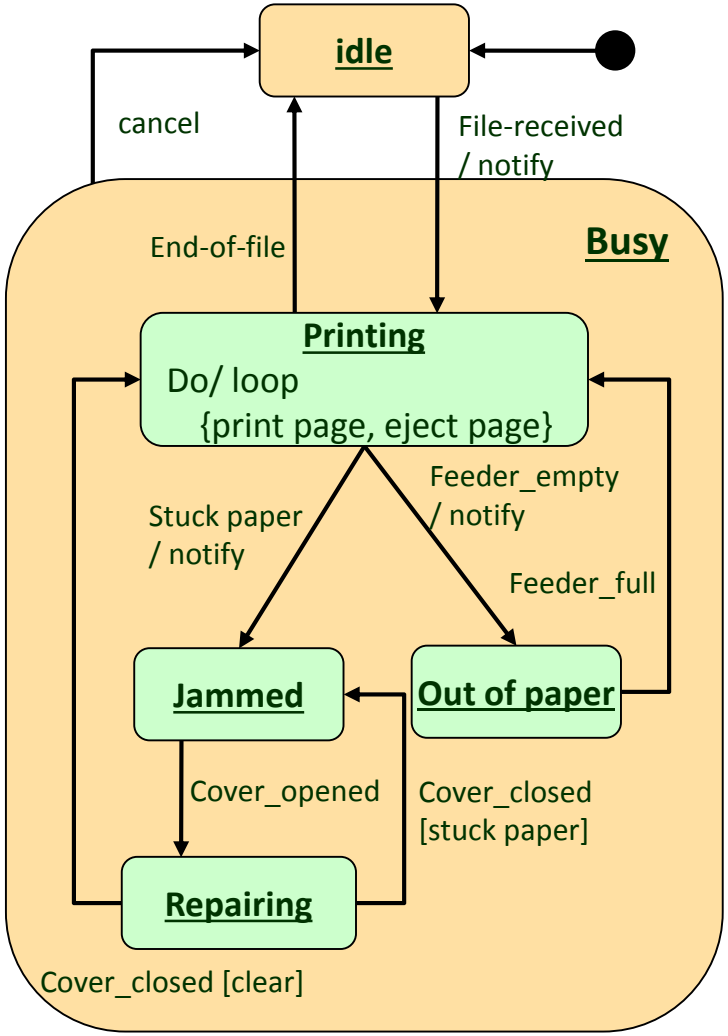
- מצב (State)
  - מצב רגיל/פסיבי (למשל: "ממתין")
  - מצב פעיל (למשל: "מעבד")
- מעבר (Transition)
  - שינוי מצב הנגרם בעקבות אירוע או תנאי
- אירוע (Event)
  - גורם למעבר ממצב למצב (למשל: "הדלקת מתג")
- תנאי (Guard)
  - מתנה את השפעת האירוע (למשל: "הדלקת מתג [הדלת סגורה]")
- פעולה (Action)
  - מתרחשת בעת מעבר או בתוך מצב
    - בכניסה למצב (**entry/** action)
    - ביציאה ממצב (**exit/** action)
    - במהלך שהִיה במצב פעיל (**do/** action)



דלת מעלית – state chart

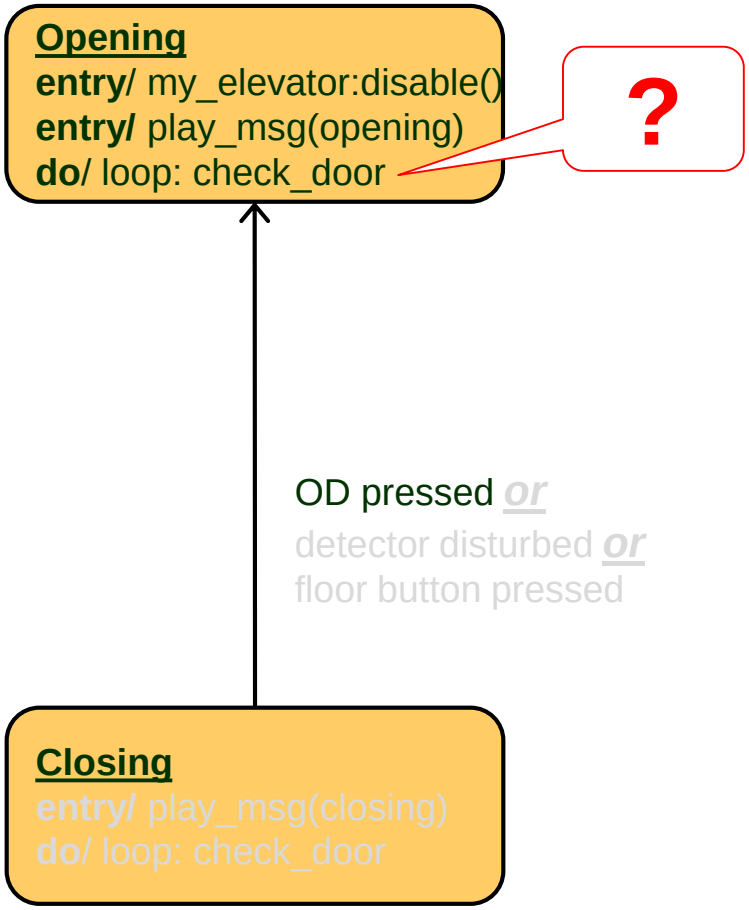


מצבים מורכבים / משולבים – דוגמה: מדפסת



דלת מעלית – Event Handler (קטע)

```
void EventHandler(event EventClass)
...
switch (event):
{   case OD_Pressed
      if (current_state==Closing)
      {
         my_elevator.disable();
         play_msg("opening");
         ...
         current_state=Opening;
      }
 break;
```

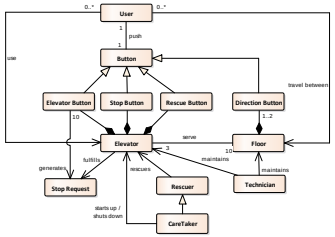


מטלת כיתה

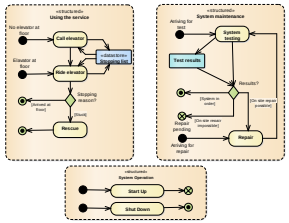
- ערכו State-Machine Diagram למחלקה Device
  - על בסיס סיפור הלקוח זהו את המצבים השונים בהם יכול להיות מתקן
  - הגדירו אירועים למעבר בין המצבים השונים
  - הוסיפו פעולות /entry , /exit , /do בתוך המצבים, על פי הצורך

כל התורה על רגל אחת...

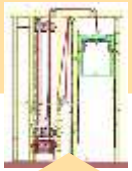
מבנה

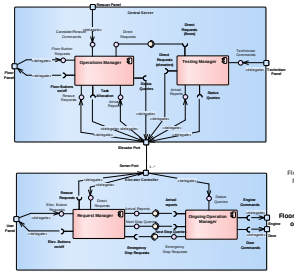


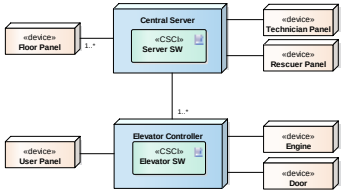
התנהגות



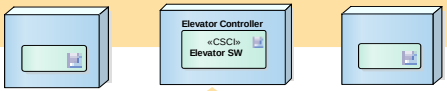
מערכת

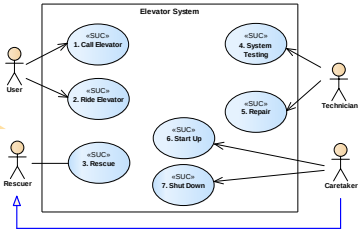




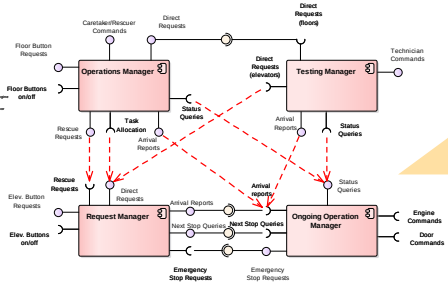


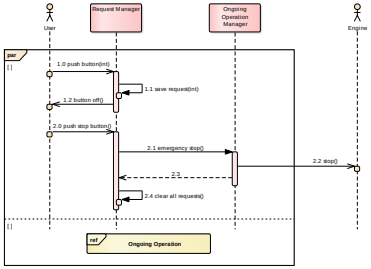
פריטים





רכיבים





יחידות