תרגיל מס' 3 – מערכת לניהול מוסך

מטרות

- הטמעה של עבודה עם מחלקות, **הורשה ופולימורפיזם**
- שימוש ב- Collections
- מימוש enums
- פיתוח ושימוש ב- Dll (אסמבלי) חיצוני
- עבודה עם מספר פרויקטים
- עבודה עם Exceptions

ידע נדרש

- תיכנות מונחה עצמים תוך שימוש בפולימורפיזם ב- #C
- שימוש ב- Collections
- פיתוח ושימוש ב- Dll (אסמבלי) חיצוני
- עבודה עם מספר פרויקטים
- עבודה עם Exceptions
- **הירושה מ- Object**

קישור לסירטון הדרכה: https://www.youtube.com/watch?v=bScwvq5-ovg

התרגיל

המטרה הסופית: מערכת קטנה ש"מנהלת" מוסך.
המערכת תדע לנהל מוסך שמטפל כיום בחמישה סוגי רכבים –

- אופנוע רגיל
  (2 גלגלים עם לחץ אוויר מקסימאלי **30**, דלק מסוג Octan98, **5.8** ליטר טנק דלק)
- אופנוע חשמלי
  (2 גלגלים עם לחץ אוויר מקסימאלי **30**, זמן מצבר מקסימאלי – **2.3** שעות)
- מכונית רגילה
  (4 גלגלים עם לחץ אוויר מקסימאלי **29**, דלק מסוג Octan95, **48** ליטר טנק דלק)
- מכונית חשמלית
  (4 גלגלים עם לחץ אוויר מקסימאלי **29**, זמן מצבר מקסימאלי – **2.6** שעות)
- משאית
  (16 גלגלים עם לחץ אוויר מקסימאלי **25**, דלק מסוג Soler, **130** ליטר טנק דלק)

לכל כלי רכב יש את התכונות:

- שם דגם (string)
- מספר רישוי (string)
- אחוז האנרגיה שנותרה במקור האנרגיה שלו (בשביל מד הדלק/חשמל) (float)
- אוסף של גלגלים
  לכל גלגל יש את התכונות הבאות:
  - שם יצרן (string)
  - לחץ אוויר נוכחי (float)
  - לחץ אוויר מקסימאלי שקבע היצרן (float)
  - פעולת ניפוח (מתודה שמקבלת נתון לגבי כמה אוויר להוסיף לגלגל, ומשנה את מצב לחץ האוויר אם הוא לא חורג מהמקסימום)

לאופנוע (רגיל/חשמלי), בנוסף לתכונות של רכב, יש גם את התכונות:

- סוג רשיון: A, A2, AA, B
- נפח מנוע בסמ"ק (int)

למכונית (רגילה/חשמלית) , בנוסף לתכונות של רכב, יש גם את התכונות:

- צבע (הצבעים האפשריים הם: אדום, לבן, שחור, כחול)
- כמות דלתות (2, 3, 4, או 5)

למשאית, בנוסף לתכונות של רכב, יש גם את התכונות:

- האם מסיע תכולה בקירור (bool)
- נפח מטען - float

בכלי רכב שעובד על דלק ניתן למצוא את המידע הבא ולהפעיל את הפעולות הבאות:

- סוג הדלק (Octan98, Octan96, Octan95, Soler)
- כמות הדלק הנוכחית בליטרים (float)
- כמות הדלק המקסימאלית בליטרים (float)
- פעולת תדלוק (מתודה שמקבלת כמות ליטרים <u>להוספה</u> וסוג דלק, ומשנה את מצב הדלק במידה וסוג הדלק מתאים ואין חריגה מגודל הטנק)

בכלי רכב חשמלי ניתן למצוא את המידע הבא ולהפעיל את הפעולות הבאות:

- זמן מצבר שנותר בשעות (float)
- זמן מצבר מקסימאלי בשעות (float)
- פעולת טעינת מצבר (מתודה שמקבלת נתון שהוא מספר שעות <u>להוסיף</u> למצבר ו"טוענת" את המצבר בהתאמה כל עוד מספר השעות לא חורג מהמקסימום)

על כל רכב שנמצא במוסך יש לשמור את הנתונים הבאים:

- שם בעלים (string)
- טלפון בעלים (string)
- מצב הרכב במוסך (המצבים האפשריים: בתיקון, תוקן, שולם)
  - כל רכב שנכנס למוסך מצבו ההתחלתי הוא 'בתיקון'.

<u>המערכת תספק את הפונקציונאליות הבאה למשתמש בה:</u>

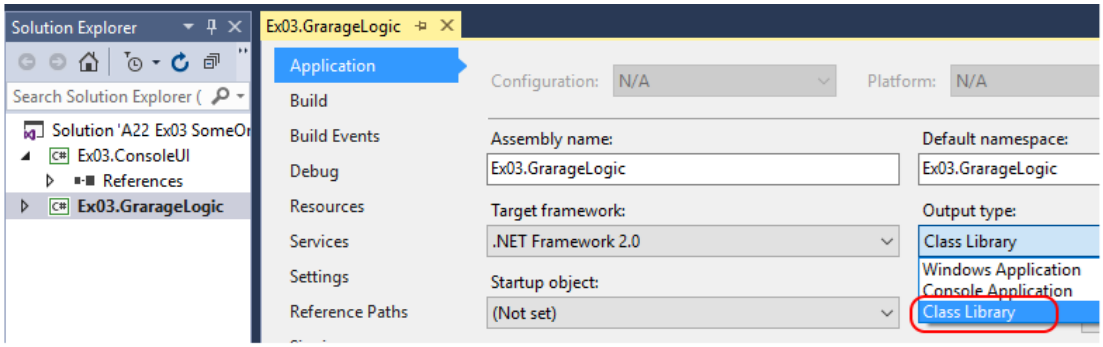
1. "להכניס" רכב חדש למוסך. אם מנסים להכניס רכב שכבר נמצא במוסך (עפ"י מספר רישוי), המערכת תוציא הודעה מתאימה ותשתמש ברכב שכבר נמצא במוסך (ותעביר את מצב הרכב ל- "בתיקון")
2. להציג את רשימת את מספרי הרישוי של הרכבים במוסך, עם אפשרות לסינון לפי המצב שלהם במוסך.
3. לשנות מצב של רכב במוסך (הנתונים המבוקשים מהמשתמש הם מספר רישוי והמצב החדש).
4. לנפח אוויר בגלגלים של רכב למקסימום (לפי מספר רישוי)
5. לתדלק רכב שמונע ע"י דלק (הנתונים הם מספר רישוי, סוג דלק למילוי, כמות למילוי)
6. להטעין רכב חשמלי (הנתונים הם מספר רישוי, כמות דקות להטענה)
7. להציג נתונים מלאים של רכב לפי מספר רישוי (מספר רישוי, שם דגם, שם בעלים, מצב במוסך, פירוט הגלגלים (לחץ אוויר ויצרן), מצב דלק + סוג דלק / מצב מצבר, ושאר הפרטים הרלוונטיים לסוג הרכב הספציפי)

[המשך בדף הבא]

לצורך פיתוח המערכת, יש ליצור Solution שיכיל **שני** פרויקטים:

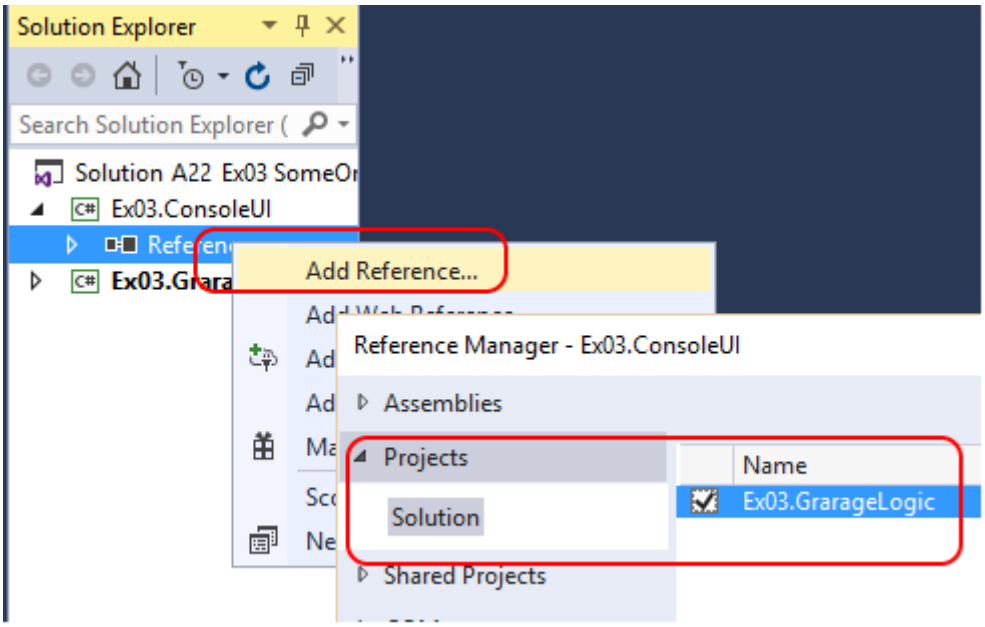
1. **Ex03.GarageLogic**

זהו פרויקט שמייצר dll (אסמבלי שאינו PE) שמכיל רק את מודל האובייקטים והשכבה הלוגית של המערכת (השכבה שנרצה לעשות בה שימוש חוזר גם אם נפתח ממשק משתמש אחר מאשר Console).
השכבה הזו אינה יודעת להדפיס למסך או לקלוט נתונים מקונסול.
כדי לייצר dll יש ללכת למאפיינים (Properties) של הפרויקט ולבחור באפשרות:
Output type: Class Library



2. **Ex03.ConsoleUI**

זהו פרויקט שמייצר את ה- exe ובו יש מימוש של ממשק המשתמש במערכת לניהול מוסך בקונסול. מערכת זו עושה שימוש במודל שמומש בפרויקט הראשון ע"י reference לפרויקט הראשון:



הערות:

1. יש לתכנן את המערכת בצורה הטובה ביותר של קוד מבחינת חלוקה למחלקות ולמתודות, הורשה ופולימורפיזם.
2. יש להפריד בין ממשק המשתמש והשכבה הלוגית אשר מנהלת את מודל האובייקטים ואת הישויות הלוגיות של המערכת (מחלקה כמו Vehicle לא אמורה להכיר את המחלקה Console, גם לא באופן עקיף). יש לנסות ולחשוב על פתרונות לצורך הפרדה זו .
3. **יש למקם את הקוד שמייצר אובייקטים (new) של מחלקות הרכבים, ורק אותו, ברכיב קוד מבודד (מחלקה נפרדת) בחלק הלוגי של המערכת (ברכיב זה מותר גם לשמור את רשימת סוגי הרכבים הנתמכים כרגע במערכת). רכיב זה אינו פונה למשתמש לא באופן ישיר ולא באופן עקיף.**
4. יש לכתוב את המערכת באופן כזה שהוספה של סוג רכב חדש (נניח "טרקטור") תוכל להתבצע **ללא שינוי בקוד** (למעט תוספת מינימאלית במחלקה מסעיף 3 שמייצרת אובייקטים)
5. יש להימנע משכפול קוד/לוגיקה.
6. **חובה** להשתמש ב:
   - List<T> **ו/או** Dictionary<K,V>
   - Enum
   - String formatting
   - FormatException (תיזרק במקרה של קלט לא מתאים ברמת ה-Parsing).
   - ArgumentException (תיזרק במקרה של קלט לא מתאים מבחינה לוגית, למשל בחירה למילוי דלק מהסוג הלא נכון)
   - ValueOutOfRangeException – **מחלקה שאתם תכתבו** (תיזרק במקרה של קלט לא מתאים מבחינה של טווח ערכים רצוי, למשל מילוי דלק מעל המותר).

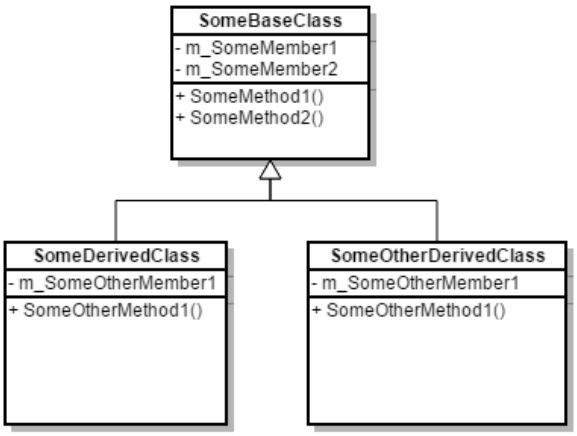
     זוהי מחלקה שיורשת מהמחלקה Exception ומחזיקה את השדות הנוספים הבאים:
     - float MaxValue
     - float MinValue

הגשה:

יש לצרף להגשה בתוך התיקייה, בנוסף לקובץ טקסט עם פרטי ההגשה, גם קובץ **word** ובו:

- שמות המגישים + ת"ז
- **רשימת הטיפוסים שהגדרתם (class/enum) עם הסבר קצרצר על כל אחד מהם.**
- **דיאגרמה שמציגה את היררכיות הירושה והקשר בין הרכיבים. לדוגמא:**

הכלי - https://www.gliffy.com/ יכול לעזור לכם לצייר את הדיאגרמה.



שם הקובץ יהיה כשם קובץ ה- solution (עם סיומת doc/docx).

בהצלחה!