תכנות מונחה עצמים מתקדם
עבודת הגשה מס ' 2

מבוא

זהו התרגיל השני בקורס ומהווה המשך לסדרת תרגילים שיינתנו במהלך הקורס, בנושא בקרת תנועה.

בתרגיל זה נממש שוב ונשדרג את המערכת שמימשנו בעבודה קודמת, והפעם נשתמש בכלים של תכנות מונחה עצמים. מומלץ להתחיל את העבודה מפרויקט חדש.

נא לקרוא את כל המסמך לפני תחילת העבודה!

<u>דגשים להגשה</u>

- ניתן להגיש עבודה זו בזוגות – רק אחד מהסטודנטים יגיש את העבודה במודל. בתיעוד בכל קובץ יש לציין שם ות.ז. של מגיש\ים, בתוך תיעוד ה javadoc.
- חלק מניקוד העבודה מתבצע ע"י בדיקות אוטומטיות ולכן חשוב מאוד להגדיר את כל המחלקות, המטודות והשדות בדיוק כפי שצוינו במסמך.
- על העבודה לעבוד עם קובץ ה GameDriver המצורף ועל הפלט להכיל את כל פרקי הפלט הנתון, כאשר החישובים והפעולות מתבצעים באותו אופן וסדר כמו בדוגמת הרצה.
- קובץ עם דוגמת הרצה נוספת Test הינו דוגמת הרצה מחייבת פרט מנתונים שמוזנים באופן רנדומלי מכוח הגדרת המחלקות. הפלט של קובץ זה נמצא בתוך הקובץ בהערות.
- לכל שאלה לפנות למתרגל האחראי על עבודה ראשונה – סופיה קרימברג במייל sofiak@ac.sce.ac.il. נושא ההודעה חייב להיות "aoop20_hw2".
- חובה לתעד כל קובץ, מחלקה ופונקציה ע"י javaDoc - ניתן להיעזר בתיעוד באתר oracle או בקבצים הרלוונטיים במודל.

<u>דגשים לעבודה זו</u>

- על כל העבודה להיות פרויקט יחיד המחולק לpackages כפי שמוגדר בהמשך.
- <u>על כל השדות בכל המחלקות להיות פרטיים בלבד. גישה אליהם תתבצע בעזרת get/set .</u>
- <u>לכתוב בכל מחלקה את הבנאים הדרושים כדי שתעבוד המחלקה הראשית GameDriver .</u>
- על כל הקבועים להיות תחת שדות final. שימו לב, כלל ההתייחסויות במהלך הקוד לערכים ספציפיים ייעשו באמצעות גישה לקבועים הנ"ל.
- העבודה מכילה נושאים המיועדים ללמידה עצמאית.
- בכל מחלקה חובה לממש **בנוסף למתודות המתוארות**: getters&setters, equals(), toString().
- חובה להקפיד על המבנה הנתון, כל הנחיה להורשה/ממשק/enum היא מחייבת.

המשימה:

חלק א'

עליכם לבצע את השלבים הבאים בעבודה זאת:

1. לממש את המערכת הנתונה בדיוק כפי שהיא מתוארת בהמשך. אין להוסיף שדות, בנאים, מחלקות וממשקים. מותר (אבל אין בכך צורך) להוסיף מתודות עזר פרטיות (private) בלבד.

חלק ב' - בונוס.

חלק זה אינו חובה ומימוש שלו מזכה בעד 40 נקודות בונוס לציון העבודה.

2. ניתן לשפר מערכת זו ע"י הוספת הורשה בין שניים מהמרכיבים הקיימים (רק ע"י שינוי הגדרות בהכרזת מחלקות/ממשקים). יש לבצע שינוי זה.
3. ללא שינוי של אף חלק מהמערכת הנתונה פרט לזה שבסעיף 2, יש ליצור package חדש ובו לממש מודול נוסף במערכת, העונה לדרישות הבאות:
   - למערכת נוסף אלמנט חדש – לכל רכב יש נהג.
   - ישנם 3 סוגי נהגים: נהג רגיל, נהג חדש ועבריין תנועה. כאשר כל אחד נוהג באופן שונה:
     - נהג רגיל נוהג כפי שהוגדר במערכת המקורית (ללא שינוי)
     - לנהג חדש נוסע בכבישים במהירות נמוכה ב-10 קמ"ש מהמהירות בה היה נוסע נהג רגיל על אותה דרך ובאותו סוג רכב. בצמתים ללא רמזור לנהג חדש לוקח זמן ארוך יותר להבין את חלוקת זכויות הקדימה בצומת, לכן הוא מתעקב בצומת מסוג זה פעימת זמן נוספת.
     - עבריין תנועה מתעלם מהגבלות מהירות של כבישים. בצמתים הוא עוקף את הרכבים שנמצאים בתור לפניו. ובצומת ללא רמזור הוא מתעלם מזכויות קדימה ועובר את הצומת ללא בדיקת עדיפויות.
   - המודול החדש יכיל קובץ הרצה משלו ולא יפעל כאשר מריצים את התוכנית ע"י קובץ הרצה המקורי הנתון.

בחלק זה ניתן להשתמש בכל הכלים שלמדתם ללא הגבלה. יש לתכנן תוסף זה תוך כדי הקפדה על עקרונות תכנות מונחה עצמים.

רכיבים למימוש במערכת הבסיסית:

1. **package utilities**

1.1 Enum VehicleType

```
public enum VehicleType {

        car(90), bus(60), bicycle(40), motorcycle(120), truck(80),
        tram(50), semitrailer(85);

        private int averageSpeed;

        VehicleType(int speed) {
           averageSpeed=speed;

        }

        public int getAverageSpeed() {
        return averageSpeed;
        }


}
```

יש ליצור את הרכיב עם הקוד הנתון ללא שינויים.

1.2 GameDriver

```
public class GameDriver {

        public static void main(String[] args) {
                Driving driving=new Driving(10, 20);
                driving.drive(20);

        }

}
```

יש ליצור את הרכיב עם הקוד הנתון ללא שינויים. קוד זה מבצע הרצה של המערכת למשך 20 פעימות (בכל מקום בו מוזכרות יחידות זמן, מדובר בפעימות). המערכת מורצת עם 10 צמתים ו-20 רכבים.
פלט של הרצה זו מצורף בקובץ נפרד.

1.3 Abstract Class Point <u>implements Utilities</u>

<u>Fields:</u>

- minVal: int

  ערך מינימלי עבור קואורדינטות X ו-Y, מאותחל ל-0 ואינו משתנה.
- maxX: int

  ערך מקסימלי עבור X, מאותחל ל-800 ואינו משתנה.
- maxY: int

  ערך מקסימלי עבור Y, מאותחל ל-600 ואינו משתנה.
- x: int
- y: int

  קואורדינטות של נקודנה במערכת צירים קרטזית.

<u>Constructors:</u>

- Point(double x, double y);

  יוצר נקודה עם קואורדינטות נתונות, במקרה והערכים אינם תקינים, מחליף אותם בערכים תקינים אקראיים.
- Point();

  יוצר נקודה עם ערכים אקראיים.

<u>Methods:</u>

- calcDistance(Point other);

  מחזירה מרחק בין הנקודה הנוכחית לנקודה שמתקבלת כארגומנט.

1.4 Interface Timer.

- incrementDrivingTime() : void

1.5 Interface Utilities.

ממשק זה מממש (!) מתודות דיפולטיביות. המחלקות המממשות אינן חייבות לממש את המתודות, אלא יכולות להשתמש במימושים שבתוך הממשק (כל עוד המתודות מסומנות ב-default).

- **default** checkValue(double Val, double min, double max) : boolean
  בודקת תקינות עבור ערכים שחייבים להיות בגבולות מינימום ומקסימום נתונים.
- **default** correctingMessage(double wrongVal, double correctVal, String varName): void
  מדפיסה הודעה על החלפת ערך לא תקין בערך תקין. ניתן לראות בדוגמת הרצה מצורפת.
- **default** errorMessage(double wrongVal, String varName): void
  מדפיסה הודעה על ערך לא תקין.
- **default** getRandomBoolean(): boolean
  מחזירה ערך בוליאני אקראי.
- **default** getRandomDouble(double min, double max): double

מחזירה double אקראי בגבולות min,max

- **default** getRandomInt(int min, int max) :double
  מחזירה מספר שלם אקראי בגבולות מינימום ומקסימום.
- **default** getRandomIntArray(int min, int max, int arraySize) : ArrayList<Integer>
  מחזירה מערך/רשימה בגודל מבוקש של מספרים שלמים אקראיים הנעים בין מינימום למקסימום.
- **default** successMessage(String objName) : void
  מדפיסה הודעה על יצירה מוצלחת של אובייקט.

2. <u>package: components</u>

2.1 Class Driving implements Utilities, Timer

מחלקה שמחברת את חלקי המערכת ומפעילה אותה.

<u>Fields:</u>

- map: Map
  מפה עבור הפעלה נוכחית
- vehicles : ArrayList<Vehicle>
  כלי רכב, שמשתתפים בהרצה
- drivingTime : int
  משתנה ששומר (צובר) זמן/פעימות מרגע התחלת ההרצה
- allTimedElements : ArrayList<Timer>

משתנה ששומר יחד את כל האלמנטים המושפעים מפעימות (רכבים ורמזורים)

<u>Constructors:</u>

- Driving (int numOfJunctions, int numOfVehicles);
  בנאי אקראי, מקבל כמות צמתים וכמות רכבים רצויה.
  יוצר מפה עם כמות צמתים נתונה, יוצר את הכמות הדרושה של רכבים ומאתחל את כל השדות.

<u>Methods:</u>

- drive(int numOfTurns) : void

מתודה מקבלת את כמות הפעימות ועבור כל פעימה מפעילה את המתודה incrementDrivingTime().

- incrementDrivingTime(): void
מתודה שמקדמת את הזמן (פעימות) עבור כל האובייקטים המושפעים מכך (רכבים ורמזורים).

2.2 Class Junction <u>extends Point implements RouteParts</u>

מייצגת צמתים ללא רמזורים. בעבודה זאת אנחנו מתייחסים לצמתים, דרכים ומסלולים כלרכיבי דרך ומפעילים עליהם מתודות דומות, המרוכזות בממשק.

<u>Fields:</u>

- Int objectsCount;
משתנה של מחלקה, מונה את כמות האובייקטים שנוצרו, מאותחל ל-1.
- ArrayList<Road> enteringRoads;
רשימת דרכים נכנסות לצומת (דרכים שמסתיימות בצומת זה)
- ArrayList<Road>exitingRoads;
רשימת דרכים יוצאות לצומת (דרכים שמתחילות מצומת זה)
- String junctionName;
שם מזהה של אובייקט

<u>Constructors:</u>

- Junction();
בנאי רנדומלי, יוצר צומת בנקודה רנדומלית, מאתחל את כל השדות, קובע את שם האובייקט לפי המספר הסידורי שלו במחלקה.
- Junction(String junctionName, double x, double y);
יוצר צומת עם פרמטרים נתונים.

<u>Methods:</u>

- addEnteringRoad(Road road): void
מתודה להוספת דרך לרשימת דרכים נכנסות
- addExitingRoad(Road road):void
מתודה להוספת דרך לרשימת דרכים נכנסות
- calcEstimatedTime(Object obj): double
מתודה המקבלת כארגומנט מופע של כלי רכב, מחשבת ומחזירה את הזמן המירבי שהרכב יצטרך להמתין לפני שיעבור את הצומת. כיוון שמדובר בצומת ללא רמזור, זמן משוער מחושב לפי כמות דרכים עם עדיפות גבוהה יותר יחסית לדרך ממנה מגיע הרכב לצומת. (עדיפות מחולקת לפי מיקום (אינדקס) במערך הדרכים הנכנסות, ככל שהאינדקס קטן יותר, העדיפות גבוהה יותר). פונקציה מחזירה את כמות הדרכים עם עדיפות גבוהה יותר מזאת שממנה מגיע הרכב פלוס 1 (חציית צומת מתבצעת בפעימה 1, לכן ההוספה).
- canLeave(Vehicle vehicle): boolean
מתודה של ממשק, שמחזירה ערך בוליאני המשקף את האפשרות לבצע צ'ק-אואוט מרכיב דרך. במקרה של צומת המתודה תחזיר תשובה האם הרכב יכול לחצות את הצומת ולעזוב אותה.
- checkAvailability(Vehicle vehicle) : boolean
מתודת עזר למתודה קודמת, בודקת אם קיימות דרכים יוצאות לצומת וגם את הכביש, ממנו הרכב הגיע לצומת: אם ברשימת המתנה של הכביש הרכב אינו הראשון בתור, יצטרך להמתין עד שרכבים שלפניו יצאו מהצומת.
- checkIn(Vehicle vehicle): void
מתודה ש"רושמת" את הרכב בצומת, מעדכנת את כל השדות הרלוונטיים של הצומת והרכב ומדפיסה על כך הודעה. המתודה שייכת לממשק וממומשת בכל מחלקה מממשת באופן שונה.

- checkOut(Vehicle vehicle) : void
  מתודת "יציאה" מרכיב דרך, שייכת לממשק רכיבי דרכך. מופעלת בתנאי שהרכב "מורשה" לחצות ולעזוב את הצומת, מעדכנת את כל השדות הרלוונטיים ומדפיסה הודעה.
- findNextPart(Vehicle vehicle) : RouteParts

מתודת ממשק, מחפשת את הרכיב הבא במסלול (לטובת בניית מסלול רנדומלי למשל) שמתאים לרכב נתון, בודקת אם קיימות דרכים יוצאות פעילות (enabled) שמאפשרות נסיעה בסוג הרכב הנתון ומחזירה את אחת מדרכים שנמצאו (אקראית). אם לא מוצאת דרך מתאימה מחזירה null.

- stayOnCurrentPart(Vehicle vehicle) : void
  מתודת ממשק, מופעלת כאשר הרכב הנתון לא יכול לעבור לרכיב דרך הבא בפעימה הנוכחית. (בעיית עדיפות, רכבים קודמים). מדפיסה הודעה מתאימה.

2.3 Class LightedJunction <u>extends Junction</u>

מחלקה עבור צמתים מרומזרים. מרחיבה את מחלקת צמתים.

<u>Fields:</u>

- TrafficLights lights;
  משתנה עבור מופע של מחלקת רמזורים, הפועל בצומת זה.

<u>Constructors:</u>

- LightedJunction()
  בנאי רנדומלי, יוצר צומת רנדומלית ללא רמזור ומוסיף אליה רמזור (לאחר שמגריל את סוג הרמזור – אקראי או עקבי)
- LightedJunction(string name, double x, double y, boolean sequential, boolean lightsOn)
  בנאי קונקרטי, המקבל כארגומנטים את הפרטים של הצומת והרמזור.

<u>Methods:</u>

- calcEstimatedTime(Object obj) : double

מתודת ממשק, מחשבת את זמן העיקוב המשוער בצומת זה. זמן משוער בצומת מרומזר מחושב לפי זמן השהיית הרמזור (delay) שמוכפל בכמות הדרכים הנכנסות בצומת כאשר מכמות זאת מוחסרת דרך אחת (הדרך שממנה הרכב מגיע). בנוסף, מוסיפים למספר שהתקבל 1 עבור חציית הצומת.

- canLeave(Vehicle vehicle) : boolean

כמו במחלקת אב, אבל כאן אפשרות היציאה תלויה ברמזור ולא בעדיפות. אם הרמזור מופעל.

2.4 Class Map <u>implements Utilities</u>

מחלקת מפות עליהן מתנהלת התנועה.

<u>Fields</u>

- ArrayList<Junction> junctions
  כל הצמתים שבמפה
- ArrayList<Road> roads
  כל הדרכים שבמפה
- ArrayList<TrafficLights> lights

כל הרמזורים

Constructors:

- Map (int numOfJunctions)
  בנאי שמייצר כמות נתונה של צמתים אקראיים ולאחר מכן יוצר דרכים בין כל צומת לכל הצמתים האחרים.
  הבנאי מגריל את סוג הצומת שיצור (מרומזר או לא מרומזר), יוצר את הדרכים ובחלק מהצמתים המרומזרים (שנבחרים גם רנדומלית) מדליק את הרמזורים.

Methods

- SetAllRoads() : void

מתודה שיוצרת את הדרכים עבור הבנאי

- turnLightsOn() : void

מתודה שמדליקה את הרמזורים עבור הבנאי.

2.5 Class RandomTrafficLights extends TrafficLights

מחלקה עבור רמזור אקראי. רמזור אקראי מאיר בירוק לדרך רנדומלית מתוך הדרכים הנכנסות של הצומת.

Fields (none)

Constructors

- RandomTrafficLights (ArrayList<Road> roads)

Methods

- changeIndex(): void

מתודה מחזירה אינדקס עבור דרך שתקבל אור ירוק (דורסת מתודה של מחלקת רמזורים).

2.6 Class SequentialTrafficLights extends TrafficLights

מחלקה עבור רמזור עקבי. רמזור זה עובר על הדרכים הנכנסות של הצומת לפי הסדר בו הן מופיעות ברשימת הדרכים ובכל פעם מדליק אור ירוק לדרך הבאה בתור. לאחר delay מכבה את האור הירוק בדרך הנוכחית ומדליק אותו בדרך הבאה ברשימה.

Fields

- int increment

מקדם סדרתי, מאותחל ל-1. אם משנים ערך זה ל-X, הרמזור יעבור בכל פעם לדרך הנמצאת במערך במרחק X מהדרך הנוכחית בה דולק ירוק. (לא נשתמש באופציה זאת בעבודה הנוכחית).

Constructors

- SequentialTrafficLights (ArrayList<Road> roads)

Methods

- changeIndex() : void

מתודה שמחזירה אינדקס חדש עבור אור ירוק.

2.7 Abstract Class TrafficLights <u>implements Timer, Utilities</u>

מחלקה אבסטרקטית עבור רמזור.

<u>Fields</u>

- int objectsCount

  מונה אובייקטים של המחלקה
- int delay

  זמן השהייה (בפעימות) שעובר בין כל החלפת אורות ברמזור. זמן ההשהייה שווה ל-0 כשהרמזור כבוי וברגע הדלקת רמזור זמן זה מוגרל ונע בין 2 ל-6.
- int greenLightIndex

  אינדקס של הדרך ברשימת הדרכים הנכנסות, לה דולק כרגע אור ירוק.
- int id

  שם מזהה למופע
- int minDelay

  ערך מינימום לזמן השהייה, מאותחל ל-2 ולא משתנה.
- int maxDelay

  ערך מקסימום לזמן השהייה, מאותחל ל-6 ולא משתנה
- ArrayList<Road> roads

  דרכים נכנסות של הצומת בו מוצב הרמזור.
- boolean trafficLightsOn

  משתנה עבור מצב דלוק/כבוי של הרמזור
- int workingTime

  מונה פעימות מרגע הדלקת הרמזור

<u>Constructors:</u>

- TrafficLights(ArrayList<Road> roads)

  בנאי מקבל רשימת דרכים נכנסות של הצומת, מאתחל את כל השדות.

<u>Methods:</u>

- abstract changeIndex() : void
- changeLights() : void

  מתודה מבצעת החלפת אורות ברמזור. מקבלת אינדקס של הדרך הבאה בתור לקבל אור ירוק, מוודאה שכל שאר הדרכים מקבלות אור אדום, מדליקה אור ירוק בדרך החדשה ומדפיסה הודעה מתאימה.
- incrementDrivingTime() : void

  מתודה שמקדמת את זמן הפעולה של הרמזור וגם בודקת אם הגיע הזמן לבצע החלפת אורות.

2.8 Class Road <u>implements RouteParts, Utilities</u>

מחלקת דרכים.

<u>Fields:</u>

- Int [] allowedSpeedOptions

מערך של ערכים קבועים אפשריים עבור מהירות מותרת בכביש. הערכים במערך הם:
30,40,50,55,60,70,80,90

- boolean enable

אם המשתנה הוא true, הדרך מופיעה על המפה ורכבים יכולים להשתמש בה.

- Junction startJunction

הצומת ממנו הדרך יוצאת

- Junction endJunction

הצומת אליו הדרך נכנסת

- boolean greenlight

משתנה עבור אור ירוק לכביש זה, מתעדכן בעת החלפת אורות ברמזור.

- double length

אורך הדרך, מחושב לפי מרחק בין שני הצמתים באמצעות נוסחת מרחק בין שתי נקודות.

- int maxSpeed

מהירות מקסימלית המותרת בדרך. מאותחל רנדומלית מתוך רשימת ערכים מותרים.

- VehicleType[] vehicleTypes

מערך של סוגי רכבים להם מותר לנסוע בדרך זאת.

- ArrayList<Vehicle> waitingVehicles

רשימת רכבים שסיימו את הנסיעה בדרך זאת וממתינים בצומת כדי לחצות אותה.

Constructors:

- Road (Junction start, Junction end)

בנאי, מאתחל את כל השדות.

Methods:

- addVehicleToWaitingVehicles(Vehicle vehicle) : void

מתודה מוסיפה רכב לרשימת רכבים ממתינים.

- calcEstimatedTime(Object obj) : double

זמן משוער בו הרכב יעבור את דרך. מתודה מקבלת את הרכב כארגומנט ומחזירה מספר מעוגל למספר שלם (!) שמתקבל מעיגול התוצאה של חילוק אורך הדרך במהירות המינימלית בין מהירות המותרת בדרך זו למהירות ממוצעת של הרכב.

- calcLength() : double

מחשבת את אורך הדרך.

- canLeave(Vehicle vehicle) : boolean

מתודה בודקת אם רכב יכל לסיים את נסיעתו בדרך זאת. מחזירה אמת כאשר הזמן שהייה של הרכב ברכיב זה שווה או גדול מזמן משוער שאמור לקחת לרכב לעבור את הדרך הזאת.

- checkIn(Vehicle vehicle) : void

מתודה "מכניסה" את הרכב לכביש, מעדכנת את כל השדות הרלוונטיים ומדפיסה הודעה מתאימה.

- checkout(Vehicle vehicle) : void

מתודה "משחררת" את הרכב מהדרך, מעדכנת את כל השדות הרלוונטיים ומדפיסה הודעה מתאימה.

- findNextPart(Vehicle vehicle) : RouteParts

מחזירה את צומת הסיום של הדרך.

- removeVehicleFromWaitingVehicles(Vehicle vehicle) : void

מסירה רכב שחצה צומת (עזב את הצומת) מרשימת רכבים ממתינים בדרך שממנה הרכב הגיע לצומת.

- stayOnCurrentPart(Vehicle vehicle) : void

מופעלת כאשר בפעימה הנוכחית רכב נשאר בדרך זאת (לא סיים את הנסיעה למשל). מדפיסה הודעה מתאימה.

2.9 Class Route <u>implements RouteParts</u>

מחלקת מסלול. מסלול בעבודה זאת הוא "רכיב דרך" וממש את הממשק של רכיבי דרך.

<u>Fields:</u>

- ArrayList<RouteParts> RouteParts
  רשימת רכיבי דרך (צמתים ודרכים) מהם מורכב המסלול, לפי סדר הנסיעה. מסלול תמיד מתחיל מדרך ותמיד מסתיים בצומת.

- Vehicle vehicle
  רכב עבורו המסלול נבנה.

<u>Constructors:</u>

- Route(RouteParts start, Vehicle vehicle)
  בנאי רנדומלי, מחפש רכיב דרך אפשרי הבא מהנקודה הנוכחית, עד שמגיע לאורך של 10 רכיבים במסלול או עד שמגיע לצומת ללא דרכים יוצאות שמתאימות לרכב זה.

<u>Methods:</u>

- calcEstimatedTime(Object obj): double
  מתודה מחשבת זמן משוער לביצוע המסלול עבור רכב זה. מקבלת רכב כארגומנט.
- canLeave(Vehicle vehicle) : boolean
  מחזירה אמת אם רכב הגיע לרכיב האחרון במסלול.
- checkIn(Vehicle vehicle) : void
  רושמת את הרכב במסלול (ברגע שהרכב מקבל את המסלול), מעדכנת את השדות הרלוונטיים ומדפיסה הודעה מתאימה.
- checkout(Vehicle vehicle): void
  מתודה "משחררת" את הרכב מהמסלול. מדפיסה הודעה מתאימה.
- findNextPart(Vehicle vehicle): RouteParts
  אם הרכב הגיע לסיום המסלול, המתודה יוצרת עבורו מסלול חדש ובאופן הבא:
  אם בצומת הנוכחי אין דרכים יוצאות מתאימות, המסלול החדש נוצר רנדומלית מנקודת ההתחלה של המסלול הישן.
  אם יש דרכים יוצאות בצומת הנוכחי, המסלול החדש מתחיל מהדרך האחרונה בה נסע הרכב.
  אם הרכב לא סיים את המסלול, מתודה מחזירה את רכיב מסלול הבא בתור אחרי הרכיב שבו הרכב נמצא כרגע.
- stayOnCurrentPart(Vehicle vehicle) : void
  מדפיסה הודעה על כך שהרכב ממשיך במסלול הנוכחי.

2.10 Interface RouteParts <u>extends Utilities</u>

ממשק רכיבי דרך.

<u>Methods:</u>

- calcEstimatedTime(Object obj) : double
- canLeave(Vehicle vehicle) : boolean
- checkIn(Vehicle vehicle) : void
- checkout(Vehicle vehicle) : void
- findNextPart(Vehicle vehicle) : RouteParts
- stayOnCurrentPart(Vehicle vehicle) :void

2.11 Class Vehicle <u>implements Utilities, Timer</u>

מחלקה מייצגת כלי רכב.

<u>Fields:</u>

- int id

  מספר מזהה של הרכב
- VehicleType vehicleType

  סוג רכב
- Route currentRoute

  מסלול נוכחי של הרכב
- RouteParts currentRoutePart

  רכיב דרך נוכחי (צומת/דרך)
- int timeFromRouteStart

  זמן (בפעימות) שעבר מרגע תחילת מסלול
- int timeOnCurrentPart

  זמן (בפעימות) שעבר מרגע שרכב ביצע צ'ק-אין ברכיב זה.
- int objectsCount

  מונה אובייקטים של המחלקה
- Road lastRoad

  דרך אחרונה בה הרכב נסע/נוסע כרגע
- String status

  מחרוזת לשמירת סטטוס (סיבת עיקוב בצומת וכדו') לטובת הדפסות, מאותחלת ל-null

<u>Constructors:</u>

- Vehicle(Road road)

  בנאי רנדומלי,
  מאתחל את כל השדות, מספר מזהה של רכב מאותחל למספר סידורי של האובייקט שלו, סוג רכב מוגרל מתוך רשימת ערכים קבועים, מופעל בנאי רנדומלי של מסלול שמשייך את המסלול לרכב, מאתחל ומעדכן את שאר השדות.

<u>Methods:</u>

- move() : void

אם הרכב יכול לסיים ברכיב הנוכחי, מבצע בו צ'ק-אאוט ולאחר מכן צ'ק-אין ברכיב הבא של המסלול.
אחרת נשאר באותו רכיב.

- incrementDrivingTime() : void

מקדם את הזמן במסלול וזמן ברכיב נוכחי ומבצע move()

בהצלחה!