שלב 3 – מימוש ובדיקות מתודות חיתוך של קרן עם גופים גיאומטריים

הגשת השלב הזה גם תתבצע בהגשה מקוונת בתיבת ההגשה של הקישור לתג של השלב החדש

מטרות השלב:

1. המשך פיתוח הפרויקט בשיטת **Extreme Programming** בכלל ו-**TDD** בפרט, כולל שלב בדיקות, מימוש ו-**refactoring**
2. המשך תרגול בנייה מסודרת של טסטים
3. מימוש תבנית עיצוב **Composite** ועבודה עם אוספים מבוססת תבנית עיצוב **Iterator** (לולאת **for(:)**)
4. חיתוכי קרן עם גוף גאומטרי

האם עברתם על מצגת המעבדה של שלב 3? עכשיו הזמן. ועוד משהו – שם התג (**tag**) להגשת השלב הינו **PR03**.

הנחיות מפורטות מתחילות בעמוד הבא

A. מכיוון שהמוֹדֶל שלנו מִתְבסס על סריקת קרניים, נרצה להגדיר במחלקות של גופים גאומטריים מתודה שתקבל קרן ותחזיר רשימת נקודות חיתוך בין הקרן לבין הגוף:

- כל העבודה תיעשה ע"פ התהליך של **TDD**: הצהרה\כותרת => תיעוד (**javadoc**) => [עבור מתודה] יצירת בדיקות יחידה (**unit tests**) => מימוש הדרגתי תוך הרצה חוזרת ונשנית של כל הבדיקות עד שכל הבדיקות מצליחות => ארגון הקוד מחדש (**refactoring**) => הרצה חוזרת של כל הבדיקות
- בחבילת **geometries**, נוסיף ממשק (**interface**) בשם **Intersectable** והרשאה **public**, בתוכו נצהיר מתודה **findIntersections** עם החתימה הבאה:

```
List<Point> findIntersections(Ray ray);
```

  - נוסיף תיעוד בפורמט **javadoc** לממשק ולמתודה הנ"ל
- נשנה את המחלקה **Geometry** כך שמעתה הוא תממש (**implements**) את הממשק החדש
- נוסיף מימושי המתודה הנ"ל במחלקות של כל הגופים הגאומטריים **המתאימים**: מישור, משולש (לגבי מצולע\פוליגון – ראו בסוף ברשימת הבונוסים), וכדור (ספירה), לגבי צינור וגליל - ראו בסוף ברשימת הבונוסים
  - **בשלב הזה של ביצוע התרגיל כל המימושים יהיו עדיין ריקים (מחיזרים null ותו לא)**
  נ.ב.: גם אם החלטתם לממש את המתודה **במצולע**, **חובה** לממש אותה **גם** **במשולש** (גם מכיוון שהבונוס הוא **בנוסף ולא במקום** המטלה הבסיסית, אך בעיקר בגלל שבמשולש ניתן לממש את המתודה בצורה יותר מהירה מבחינת זמן ריצה – וזה יהיה משמעותי בהמשך הדרך)
  נ.ב.: בהמשך, בעת מימוש בפועל של המתודה בכל המחלקות המתאימות: במקרה שלא נמצאו נקודות חיתוך – המתודה תמיד תחזיר ערך **null**
- בכל המחלקות של בדיקות של **כל** הגופים הגאומטריים, נוסיף מתודת בדיקה עבור המתודה **findIntersections** שתממש את כל הבדיקות כפי שנלמד בקורס התאורטי
  - הבדיקות צריכות לכלול את כל מה שנלמד בהרצאה מבחינת מחקות שקילות (**EP**) ומקרי גבול וקצה (**BVA**)
  - הבדיקות חייבות לכסות **לפחות** את הבדיקות שמופיעות במצגת הקורס התיאורטי, וניתן להוסיף לפי שיקול דעתכם
  - כדאי לְקַבֵּץ את הבדיקות לקבוצות ולשים הערה המכילה תיאור קבוצה עבור כל אחת מהקבוצות, ראו דוגמה (ללא מימושים של רוב הטסטים – שעליכן להשלים) בנספח שבסוף המסמך הזה
  - **בנספח** בסוף ההנחיות יש דוגמה של בדיקות (תבנית בדיקות המתודה עבור כדור, כולל שני **test case** הראשונים) **– חובה להשתמש בדוגמה הזו כבסיס לכתיבת כל המתודות של בדיקות המתודה עבור כל הגופים**
- **לאחר מכן נממש** את המתודה בפועל, בכל המחלקות **המתאימות**, ובהתאם למודל המתמטי שהוצג בהרצאות בקורס התאורטי

מספר הערות חשובות בעניין המימושים של המתודה:

- **חובה** להשתמש במתודות **isZero** ו-**alignZero** ממחלקת **Util** עבור\לפני כל הבדיקות, למשל, כאשר אנחנו מחשבים את המכנה לנוסחה עבור המישור:

```
double nv = n.dotProduct(v);
if (isZero(nv)) …
```

  או כאשר אנחנו מחשבים את ***t*** לפי הנוסחה:

```
double t = alignZero(nQMinusP0 / nv);
if (t > 0) …
```

  נ.ב.: בדוגמאות האלה הנחנו שבתחילת המוֹדוּל מופיע **import static primitives.Util.*;**
- **אסור לכלול את ראשית הקרן** בנקודות חיתוך (על מנת למנוע סיבוכים מיותרים ובאגים קשים לדיבאג בהמשך הפרויקט)
- **אסור** ליצור רשימה לפני שיודעים בוודאות ש**יש** נקודות חיתוך **וכמה יש** נקודות חיתוך (בגלל אילוצי זמני ריצה שהמשך הדרך – יצירת אובייקטים זו פעולה כבדה הכוללת הקצאת זיכרון דינמי וכו')
- **חובה** להחזיר **null** כאשר אין נקודות חיתוך
- **חובה** ליצר רשימה עם נקודות רק פעם אחת – כאשר כל נקודות החיתוך ידועות
- **חובה** להשתמש ב-***List.of(…)*** ליצירת הרשימה המוחזרת עבור רשימות חיתוך של גוף פשוט (לא מורכב)
- כאשר גוף פשוט (לא מורכב) מייצר שתי נקודות חיתוך (כדור, גליל), הנקודות ברשימה צריכות להיות מסודרות לפי המרחק מראש הקרן), זאת אומרת עם וקטור של כיוון הקרן מצביע "ימינה", הנקודות יהיו מסודרות "משמאל לימין".
- **אסור לכלול נקודות על גבולות גופים דו-ממדיים (למשל צלעות וקודקודים של משולשים\מצולעים) ונקודות השקה** (השקה לפני ספירה, גליל) ברשימת נקודות חיתוך – (על מנת למנוע סיבוכים רבים ובאגים קשים לדיבאג בהמשך הפרויקט)
- בתכנון הבדיקות, **כדאי מאוד למקם** את הגופים ואת קרניים בבדיקות **בצורה נוחה** על מנת להקל על חישוביכם (בחישוב מקדים של התוצאות הצפויות), אך **אסור להתבסס על מרכז הקואורדינטות** (נקודה (0,0,0)), מכיוון שבגלל ייחודיותה חלק מהבדיקות תהפוכנה לבדיקות לא טובות (שלא מוצאות את הבאגים) – תזכרו את "**פרדוקס ההדברה**"

B. עד עכשיו התייחסנו לכל גוף באופן עצמאי. אך על מנת לבנות בעתיד סצנה שמורכבת ממספר גופים יש לאגד את כל הרכיבים ביחד ולשמור את אוסף הגופים שמתארים את הסצנה באגד גופים. כמו כן לפעמים נרצה להחזיק כמה גופים באגד גופים מסיבות פרקטיות, עיצוביות וביצועיות. לכן:

- בחבילת **geometries**, נוסיף מחלקה חדשה **Geometries** – עבור אגד גופים גאומטריים
  - המחלקה תממש את הממשק **Intersectable** ע״פ תבנית עיצוב **Composite**, המחלקה תכלול:
    - שדה של רשימת גופים, פרטי ולא ניתן לשינוי, מטיפוס **List<Intersectable>**, מאותחל עם רשימה מקושרת ריקה (אובייקט מטיפוס **LinkedList**) – יש לאתחל את הרשימה **בהגדרת השדה** ולא בבנאי!
    - בנאי ברירת מחדל (ריק – ללא קוד בתוך גוף הבנאי)
    - בנאי עם החתימה הבאה: **public Geometries(Intersectable... geometries)**
      - הבנאי ישתמש במתודה הבאה במקום שתיעשה את אותה העבודה פעמיים – DRY
    - מתודת הוספת גוף\גופים לאגד: **public void add(Intersectable... geometries)**
    - **<u>אין</u>** להוסיף מתודת מחיקה מהאגד, כי לא נצטרך למחוק גופים מהאגד אלא רק נוסיף אליו
  - נוסיף במחלקה את מימוש המתודה **findIntersections**
    - **כרגע** מימוש מתודה יחזיר רק **null** (״מימוש ריק״)
- בחבילת **unittests** נוסיף מחלקה של בדיקות **GeometriesTests**, הבדיקות שבתוכה תכלולנה מקרים כדלקמן (בבדיקות של אוסף לא רק יש לכלול לפחות אחד מכל סוג של גוף):
  - אוסף גופים ריק **(BVA)**
  - אף צורה לא נחתכת **(BVA)**
  - צורה אחת בלבד נחתכת **(BVA)**
  - כמה צורות (אך לא כולן) נחתכות **(EP)**
  - כל הצורות נחתכות **(BVA)**
- <u>בבדיקות הנ״ל אין צורך לבדוק את הנקודות אלא לבדוק את כמות נקודות החיתוך שהתקבלו</u>
- **<u>לאחר מכן</u>** נממש את המתודה **findIntersections** על פי הכללים של מחלקה מורכבת בתבנית עיצוב **Composite**
  - המתודה תחזיר רשימת נקודות החיתוך שכוללת את נקודות החיתוך של הקרן עם כל הגופים שבאגד
  - אם אין אף נקודת חיתוך עם אף גוף באגד – המתודה תחזיר **null**
  - **<u>אסור</u>** ליצור אובייקט של רשימה לפני שיודעים בוודאות שיש נקודות חיתוך ברשימה
    - זאת אומרת – אתחול של המשתנה עבור הרשימה (לפני הלולאה) יהיה ע״י ערך **null**
    - המשתנה יקבל ערך שונה רק בפעם הראשונה שמתברר שיש להוסיף נקודות חיתוך לרשימה הזו
- לבסוף נבצע ארגון מחדש (**refactoring**) עבור קוד החישוב של נקודה על קרן: $\boldsymbol{P} = \boldsymbol{P_0} + \boldsymbol{t} \cdot \boldsymbol{v}$
  - נוסיף במחלקה **Ray** מתודה **public Point getPoint(double t)** עבור החישוב הנ״ל
    - המתודה תוכל לקבל כארגומנט כל מספר ממשי, ובפועל תחזיר כל נקודה שעל הישר של הקרן (גם את ראש הקרן, אם $\boldsymbol{t} = \mathbf{0}$, אפשר ע״י בדיקה בעזרת **isZero** או ע״י תפיסת חריגה שנזרקה מהכפלת הווקטור $\boldsymbol{v}$ בסקלר $\boldsymbol{t}$)
    - כמובן, לא נשכח להוסיף תיעוד **javadoc** למתודה החדשה – *המתודה מחשבת נקודה על הישר של הקרן, במרחק נתון מראש הקרן*
  - נוסיף מחלקה **RayTests** בחבילה המתאימה של בדיקות מתכנת, בה נבדוק את המתודה החדשה הזו:
    - יש שתי מחלקות שקילות – עבור מרחק שלילי וחיובי
    - מקרה גבול אחד – עבור מרחק 0, כנ״ל

נ.ב. בנינו את המחלקה **Ray** לייצג קרן של ישר, אך לפעמים היא מייצגת ישר (למשל – עבור ציר של גליל אינסופי)

- בכל המימושים של **findIntersections** הרלוונטיים, נחליף את חישוב נקודות החיתוך על הקרן ע״י שימוש במתודה הזו
- גם לא נשכח שבמימוש **getNormal** **של גליל אינסופי**, גם יש קוד זהה ש**<u>חובה</u>** להחליפו ע״י שימוש במתודה הזו

נ.ב.1 בבדיקות של ספירה וגלילים יכולה להתקבל רשימה של שתי נקודות. נזכיר שהנקודות ברשימה המתקבלת ממימוש המתודה **findIntersections** בספירה ובגליל – מסודרות ע״פ מרחק מראש הקרן. לכן יש לוודא שאתם מסדרים את הנקודות ברשימה של תוצאה צפויה בבדיקה לפי סדר זה גם.

נ.ב.2 **<u>אסור</u>** להשתמש ב-**iterator** של **java** בצורה גולמית, אלא חובה להשתמש בלולאת ״**foreach**״ של **Java**, זאת אומרת – אל תיצרו בצורה מפורשת אובייקט **iterator** של הרשימה, אלא פשוט תשתמשו בלולאת ״**for(:)**״.

C. <u>בונוסים של התרגיל</u>:

- **<u>בדיקות</u>** + **מימוש** חיתוכי קרן עם מצולע (1 נק')
- **<u>בדיקות</u>** + **מימוש** חיתוכי קרן עם צינור [גליל אין-סופי] (2 נק')
  - עליכם לבנות או למצוא באינטרנט את המודל המתמטי, להבינו ולממשו. רמז: מקבלים משוואה ווקטורית ריבועית. במשוואה צריך לחשב את המקדמים של המשוואה, לחשב את הדטרמיננטה, לבדוק את המקרים ע"פ הדטרמיננטה (0 נק', 1 נק', 2 נק') ואז למצוא את הנקודות.
  - על מנת לקבל את מלוא ניקוד הבונוס – חובה לתכנן ולבנות טסטים (לפני המימוש!) – יש כאן מספר מקרים (כ-70) שיש להתייחס אליהם (מחלקות שקילות ומקרי קצה\גבול). רמז: מיקום מול ראשית קרן, זויות בין וקטורים (0 או 90 מעלות), וכדומה. הבונוס המלא יינתן רק למי שיצליח לבנות לפחות כ-40 טסטים רלוונטיים שונים
- **<u>בדיקות</u>** + **מימוש** חיתוכי קרן עם גליל [סופי] (2 נק')
  - כמובן, הבונוס יכול להתבצע רק אם עשיתם את הבונוס של צינור כנ"ל
  - נקודות חיבור בין בסיס למעטפת הגליל נכללים בחיתוכים (אלא אם מדובר במשיק)
  - אין ליצור אובייקטים מיותרים (אם תרצו – מישורים של בסיס, וכדומה) אלא רק בבנאי (ולשומרם בשדות בהתאם, תוך הקפדה כל כללי עיצוב של הפרויקט)
  - תכנון הבדיקות בבונוס הזה – הוא חלק חשוב מאד גם, ורק מימוש עם בדיקות כנדרש יזכה בבונוס מלא

ציון התרגיל יינתן:

- לפי אחוז הכיסוי (הביצוע) של משימת התרגיל – כולל בדיקות כל הפעולות
- על ההקפדה על הכללים (פרמוט [הזחות, רווחים, שורות רווח]
- על תיעוד **javadoc** למחלקות ולמתודות
- על הקפדה על מוסכמות שמות
- על יעילות ביצוע והקפדה על עקרונות דיזיין
- על הקפדה בדרישות\תבנית מימושים של דריסות המתודות **equals** ו-**toString**

<u>חובה</u> לבצע יצירת תיעוד **Javadoc** (**Generate Javadoc**) עבור כל הפרויקט, מהרשאה **private**, **<u>לעקוב</u>** אחרי כל התקלות ביצירת התיעוד **<u>ולתקן</u>** אותן. הנחיות יצירת התיעוד מופיעות בהחיות הכלליות לעבודה על כל השלבים.

<u>שימו לב</u> – אי הקפדה של ההנחיות לעיל עלולה לגרור הורדה בציון

<u>נספח</u>: הדגמה חלקית של המתודה **`testFindIntersections`** של **`SphereTests`** בעמוד הבא...

```java
/** A point used in some tests */
private final Point  p001 = new Point(0, 0, 1);
/** A point used in some tests */
private final Point  p100 = new Point(1, 0, 0);
/** A vector used in some tests */
private final Vector v001 = new Vector(0, 0, 1);

/**
 * Test method for {@link geometries.Sphere#findIntersections(primitives.Ray)}.
 */
@Test
public void testFindIntersections() {
   Sphere       sphere = new Sphere(p100, 1d);

   final Point  gp1    = new Point(0.0651530771650466, 0.355051025721682, 0);
   final Point  gp2    = new Point(1.53484692283495, 0.844948974278318, 0);
   final var    exp    = List.of(gp1, gp2);

   final Vector v310   = new Vector(3, 1, 0);
   final Vector v110   = new Vector(1, 1, 0);

   final Point  p01    = new Point(-1, 0, 0);

   // ============ Equivalence Partitions Tests ==============
   // TC01: Ray's line is outside the sphere (0 points)
   assertNull(sphere.findIntersections(new Ray(p01, v110)), "Ray's line out of sphere");

   // TC02: Ray starts before and crosses the sphere (2 points)
   final var result1 = sphere.findIntersections(new Ray(p01, v310));
   assertNotNull(result1, "Can't be empty list");
   assertEquals(2, result1.size(), "Wrong number of points");
   assertEquals(exp, result1, "Ray crosses sphere");

   // TC03: Ray starts inside the sphere (1 point)
     ...
   // TC04: Ray starts after the sphere (0 points)
     ...

   // =============== Boundary Values Tests ==================

   // **** Group 1: Ray's line crosses the sphere (but not the center)
   // TC11: Ray starts at sphere and goes inside (1 points)
   // TC12: Ray starts at sphere and goes outside (0 points)

   // **** Group 2: Ray's line goes through the center
   // TC21: Ray starts before the sphere (2 points)
   // TC22: Ray starts at sphere and goes inside (1 points)
   // TC23: Ray starts inside (1 points)
   // TC24: Ray starts at the center (1 points)
   // TC25: Ray starts at sphere and goes outside (0 points)
   // TC26: Ray starts after sphere (0 points)

   // **** Group 3: Ray's line is tangent to the sphere (all tests 0 points)
   // TC31: Ray starts before the tangent point
   // TC32: Ray starts at the tangent point
   // TC33: Ray starts after the tangent point

   // **** Group 4: Special cases
   // TC41: Ray's line is outside sphere, ray is orthogonal to ray start to sphere's center line
   // TC42: Ray's starts inside, ray is orthogonal to ray start to sphere's center line

 }
```