**שלב 3 – מימוש פונקציית חיתוך של קרן עם גופים**

לפני שממשכים לשלב הזה יש לוודא שלא שכחתם ליצור **tag** ב-**commit** האחרון עבור השלב הקודם.

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**
4. עבודה עם אוספים מבוססת תבנית עיצוב **Iterator**
5. חיתוכי קרן עם גוף גאומטרי

מכיוון שהמודל שלנו יתבסס על סריקת קרניים, נרצה להגדיר בגאומטריות פעולה שתקבל קרן ותחזיר נקודות חיתוך עם הגאומטריה.

נגדיר בחבילת **geometries** ממשק (אינטרפייס) חדש בשם **Intersectable** ובתוכו פעולה**findIntersections** עם חתימה כדלקמן:

**List<Point3D> findIntsersections(Ray ray);**

נוסיף תיעוד בפורמט **javadoc** לממשק החדש ולפעולה הנ"ל.

נשנה את הממשק **Geometry** כך שמעתה הוא יירש\ירחיב (**extends**) את הממשק החדש הנ"ל.

הפונקציה תמומש עבור כל הגופים הגאומטריים במחלקות מתאימות – במחלקות מישור, משולש (ו-\או פוליגון – ראו בסוף רשימת בונוסים), כדור, (צינור, גליל - ראו בסוף רשימת בונוסים). במקרה שלא נמצאו נקודות חיתוך – הפעולה תחזיר **null**. בשלב הזה נוסיף רק מימושים ריקים (שפשוט מחיזרים **null**).

במחלקות הבדיקות של הגופים הגאומטריים נוסיף בדיקות עבור הפעולה **findIntersections** שתממש את כל הבדיקות כפי שנלמד. הבדיקות צריכות לכלול את כל מה שנלמד בהרצאה מבחינת קטגוריות (**EP**) ומקרי קצה (**BVA**) לפי העקרונות של **TDD**. הבדיקות חייבות לכסות לפחות את הבדיקות שמופיעות במצגת המעבדה. כדאי לקבץ את הבדיקות לקבוצות ולשים הערת כותרת עבור כל קבוצה. דוגמה (ללא מימושים של רוב הטסטים – שעליכן להשלים) בנספח שבסוף המסמך הזה. חובה להשתמש בדוגמה הזו כבסיס לכתיבת כל הפעולות של בדיקות.

לאחר מכן תמומשנה כל הפעולות בפועל בכל המחלקות הרלוונטיות בהתאם למודל המתמטי שהוצג בהרצאות בקורס התאורטי ומופיע לרענון במצגת המעבדה.

מספר הערות בעניין המימושים האלה:

* חובה להשתמש בפונקציות **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 המודול מופיע **import static primitives.Util.\*;**

* **אין לכלול את ראשית הקרן** בנקודות חיתוך (מונע הסתבכויות מסוימות ובאגים קשים לדיבאג בהמשך הדרך)
* אין ליצור רשימה לפני שיודעים בודאות שיש נקודות חיתוך וכמה יש נקודות חיתוך (בגלל אילוצי זמני ריצה שהמשך הדרך – יצירת אובייקטים זו פעולה כבדה הכוללת הקצאת זיכרון דינמי וכו')
* יש להחזיר **null** כאשר אין נקודות חיתוך
* **אין לכלול נקודות על גבולות גופים דו-ממדיים (למשל צלעות וקודקודים של משולשים\מצולעים) ונקודות השקה** בנקודות חיתוך – (מונע הסתבכויות רבות ובאגים קשים לדיבאג בהמשך הדרך – אנחנו לא בונים ספריה גרפית מקיפה)
* כדאי למקם את הגופים ואת קרניים בבדיקות בצורה נוחה על מנת להקל על חישוביכם את התוצאות הצפויות, אך **אין להתבסס על מרכז הקואורדינטות** – נקודה (0,0,0) שבגלל ייחודיותה חלק מהבדיקות תהפוכנה לבדיקות לא טובות (שלא מוצאות את הבאגים).

עד עכשיו התייחסנו לכל גוף באופן עצמאי, 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 של גופים גאומטריים. המחלקה תממש את הממשק **Intersectable** ע"פ תבנית עיצוב **Composite**. המחלקה תכלול:

* שדה פרטי של רשימת גופים **List<Intersectable>**
* בנאי ברירת מחדל המאתחל את השדה הנ"ל עם רשימה ריקה (לבחירתכם **ArrayList** או **LinkedList** – עליכם להסביר את הבחירה בהצגת התרגיל)
* בנאי עם החתימה הבאה: **public Geometries(Intersectable... geometries)**
* פעולת הוספה לאוסף: **public void add(Intersectable... geometries)**
* אין צורך להוסיף פעולת מחיקה מהאוסף כי לא נמחק גופים מקבוצת הגופים אלא רק נוסיף.
* המחלקה תצטרך לממש את הפעולה **findIntersections**. בהתחלה מימוש הפעולה יחזיר רק **null** (מימוש ריק).
* בחבילת **unittests** תיבנה מחלקת בדיקות **GeometriesTests**. הבדיקות 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**)
* בבדיקות הנ"ל אין צורך לבדוק את הנקודות אלא לבדוק את כמות נקודות החיתוך שהתקבלו
* לאחר מכן נממש את הפעולה **findIntersections** על פי הכללים של מחלקה מורכבת בתבנית עיצוב **Composite**. הפעולה תחזיר אוסף נקודות חיתוך שכולל נקודות חיתוך של כל הגופים שבאוסף. אם אין אף נקודת חיתוך באף גוף – הפעולה תחזיר **null**.
* לבסוף יש לבצע ארגון מחדש (**refactoring**) עבור קוד החישוב של נקודה על קרן: . יש להוסיף במחלקה **Ray** פעולה **public ~~void~~ Point3D getPoint(double t)** עבור החישוב הנ"ל ולהשתמש בה בכל מקום שנדרש במימושים של **findIntersections** הנ"ל.

**נ.ב.** בבדיקות של ספירה וגלילים יכולה להתקבל רשימה של שתי נקודות. לכן יש צורך בלסדר את הנקודות על מנת להקל בכתיבת טסטים, כפי שאפשר לראות בדוגמה בנספח. על מנת לאפשר זאת נדרשת גישה לערכי הקואורדינטות של הנקודה. לכן בשלב הזה נוסיף למחלקה **Point3D** גטרים עבור קואורדינטות שיחזירו את ערכי הקורורדינטות (**double**). אין צורך להוסיף בדיקות לגטרים.

בונוסים של התרגיל:

* בדיקות + מימוש חיתוכי קרן ומצולע (1 נק')
* בדיקות + מימוש חיתוכי קרן וצינור [גליל אין-סופי] (2 נק')
  + עליכם לבנות או למצוא באינטרנט את המודל המתמטי, להבינו ולממשו. רמז – מקבלים משוואה (ווקטורית) ריבועית, בה 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.
  + יש 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.
* בדיקות + מימוש חיתוכי קרן וגליל [סופי] (1 נק')
  + תכנון הבדיקות בונוס הזה – הוא חלק חשוב מאד גם

בהצלחה!

נספח עם חלק מקוד הפעולה **testFindIntersections** של **SphereTests** בעמוד הבא.

/\*\*

\* Test method for {@link geometries.Sphere#findIntersections(primitives.Ray)}.

\*/

@Test

**public** **void** testFindIntersections() {

Sphere sphere = **new** Sphere(1d, **new** Point3D(1, 0, 0));

// ============ Equivalence Partitions Tests ==============

// TC01: Ray's line is outside the sphere (0 points)

*assertNull*("Ray's line out of sphere",

sphere.findIntersections(**new** Ray(**new** Point3D(-1, 0, 0), **new** Vector(1, 1, 0))));

// TC02: Ray starts before and crosses the sphere (2 points)

Point3D p1 = **new** Point3D(0.0651530771650466, 0.355051025721682, 0);

Point3D p2 = **new** Point3D(1.53484692283495, 0.844948974278318, 0);

List<Point3D> result = sphere.findIntersections(**new** Ray(**new** Point3D(-1, 0, 0),

**new** Vector(3, 1, 0)));

*assertEquals*("Wrong number of points", 2, result.size());

**if** (result.get(0).getX() > result.get(1).getX())

result = List.*of*(result.get(1), result.get(0));

*assertEquals*("Ray crosses sphere", List.*of*(p1, p2), result);

// TC03: Ray starts inside the sphere (1 point)

...

// TC04: Ray starts after the sphere (0 points)

...

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

// \*\*\*\* Group: 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: Ray's line goes through the center

// TC13: Ray starts before the sphere (2 points)

// TC14: Ray starts at sphere and goes inside (1 points)

// TC15: Ray starts inside (1 points)

// TC16: Ray starts at the center (1 points)

// TC17: Ray starts at sphere and goes outside (0 points)

// TC18: Ray starts after sphere (0 points)

// \*\*\*\* Group: Ray's line is tangent to the sphere (all tests 0 points)

// TC19: Ray starts before the tangent point

// TC20: Ray starts at the tangent point

// TC21: Ray starts after the tangent point

// \*\*\*\* Group: Special cases

// TC19: Ray's line is outside, ray is orthogonal to ray start to sphere's center line

}