שלב 2 – בדיקות אוטומטיות ומימוש חישובי נורמלים

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

שלב 2 : על בסיס חבילות הפרימיטיבים והגופים הגאומטריים שהגדרנו בשלב 1 נוכל עכשיו להרחיב את אוסף המחלקות ולהוסיף כמה פעולות. והעיקר – נתחיל לבנות מערכת בדיקות אוטומטיים – **Unit Tests** על מנת לבנות בסיס ל-**TDD** – פיתוח מבוסס בדיקות.

האם עברתם על מצגת המעבדה של שלב 2? האם עברת על הדגמות של **Junit** – עם **eclipse** ועם **IntelliJ IDEA**? עכשיו הזמן. ועוד משהו – שם התג (**tag**) להגשת השלב הינו **PR02**.

שימו לב : העתקת בדיקות מהתוכנית הראשית של שלב 1 לטסטים של השלב הזה תזכה אתכם בהערות המרצה ובהורדת הציון! כל הבדיקות חייבות לעבור התאמה לצורת העבודה עם **Junit** :

- גרסת **JUnit** הנדרשת הנה גרסה 5 (**Jupiter**)
- אין להשתמש ב-**assertTrue/assertFalse** אלא אם אין ברירה עקב מורכבות התנאי
- השוואת תוצאת פעולה לתוצאה צפויה חייבת להיות באופן ישיר בין תוצאת פעולה לבין תוצאה צפויה, למשל אין להשוות ~~***assertEquals(0, action(...) – result, ACCURACY, "Wrong action result");***~~, אלא יש לכתוב :
  assertEquals(result, action(...), ACCURACY, "Wrong action result");
- בבדיקות תוצאות מטיפוס **double** חובה להשתמש בפרמטר של דיוק הבדיקה (כנ"ל)

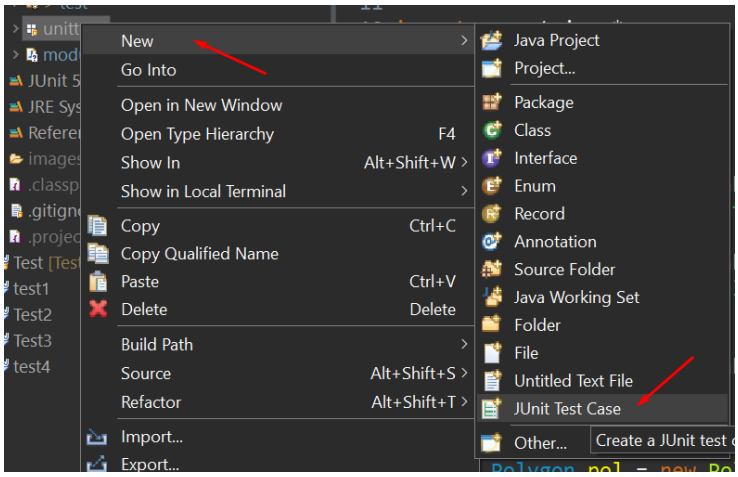
<u>ועוד מה שמומלץ לפני ביצוע שלב 2</u> : לעיין במדריך המשתמש המקורי של **Junit** :
https://junit.org/junit5/docs/current/user-guide/#writing-tests

<u>ההדגמה במסמך הזה מתבצעת על בסיס סביבת פיתוח **eclipse**</u>. המשתמשים בסביבת פיתוח **IntelliJ IDEA** או אחרת – יפעלו לפי הנחיות והקישורים הנמצאים ביחידת הוראה שמכילה את הקישורים של מר אליעזר גנסבורגר או לפי מה שימצא בעצמו ברשת.

1) נוסיף לפרויקט את ספריית הבדיקות האוטומטיות **JUnit** ע"פ ההנחיות שקיבלנו בכיתה

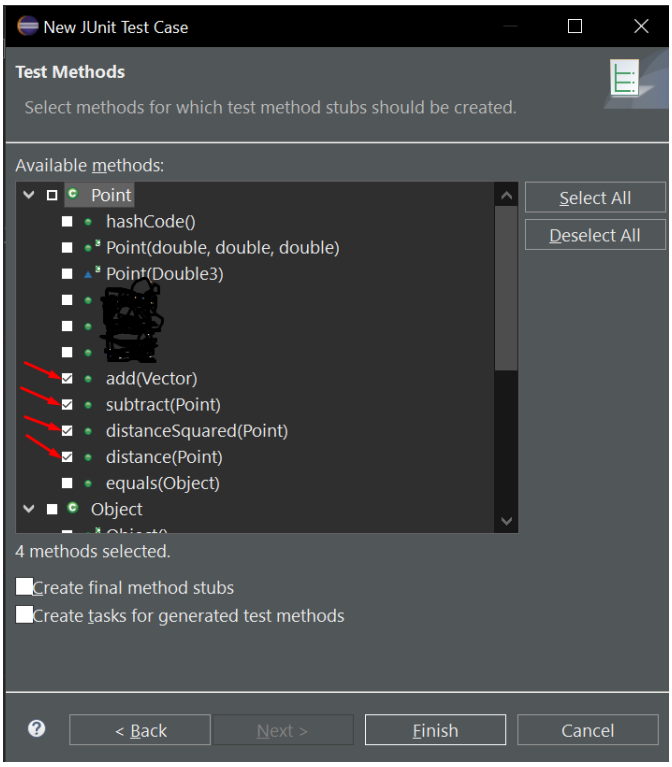
2) הכנת מחלקות הבדיקות :

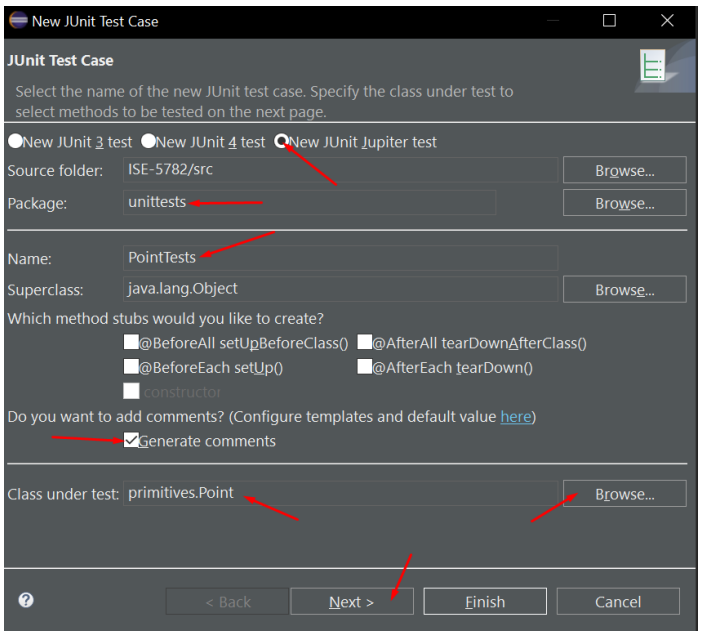
- נוסיף חבילת בדיקות (<u>טסטים</u>) **unittests**
- נוסיף תת-חבילות בדיקות **unittests.primitives** ו-**unittests.geometries**
- בתת-חבילה **unittests.primitives**, ניצור מחלקות בדיקה **PointTests** ו-**VectorTests** כפי שמודגם בהמשך
- בתת-חבילה **unittests.geometries**, ניצור מחלקות בדיקה עבור כל גוף גאומטרי בהתאם (**PlaneTests**, וכו')
- ***נ.ב.*** *שימו לב שבצעד הזה עדיין **<u>אסור</u>** לממש את הפעולות **getNormal(Point)** במחלקות הגופים הגאומטריים*
- על מנת ליצור מודול (מחלקה) לבדיקת מחלקה מסוימת מהפרויקט נעמוד עם העכבר מעל שם החבילה **unittests**, נפתח תפריט הקשר (לחיצה על כפתור ימין של העכבר) ונוסיף משם בדיקה חדשה :



- בחלון שנפתח נוודא שהבדיקה מסוג **Junit Jupiter**, נרשום את שם המחלקה של בדיקות, נסמן יצירת הערות ונרשום את השם המלא או נבחר בעזרת הכפתור **Browse...** את המחלקה הנבדקת ונלחץ על הכפתור **Next>** :

- בחלון הבא נסמן את הפעולות שאנחנו רוצים לבדוק ונלחץ על הכפתור **Finish**:





- **נ.ב.** חובה לבדוק פעולת חיסור וקטורים (למרות שבפועל לא דרסנו אותה במחלקת **Vector**) – הסיבה לכך שכאשר אנחנו כותבים בדיקות – אנחנו עוד לא יודעים (או במקרה שלנו – מתעלמים) מדרך המימוש שלנו!
- נעשה את הפעולות כנ"ל עבור כל מחלקת בדיקות
- שימו לב (גם אם בחלק מההדגמות זה נראה אחרת) – מחלקות ומתודות של בדיקות לא חייבות להיות **public** אך אסור שיהיו **private**, יחד עם זאת מומלץ לרשום שום הרשאת גישה (זאת אומרת – להשאיר הרשאת גישה ברירת מחדל - **package friendly**)
- בשלב הבא קודם כל נשלים את תיעוד ה-**javadoc** של המחלקה והמתודות כנדרש, ושימו לב שהקישור (שמופיע בתיעוד – **@link**) למחלקה או למתודה הנבדקת חייב להיות תקין:

```
package Po

/**
 * Unit tests for primitives.Point class
 * @author Yossi Cohen
 */
class PointTests {
    /**
     * Test method for {@link primitives.Point#add(primitives.Point)}.
     */
    @Test
    void testAdd() {
        fail("Not yet implemented");
    }
```

- כשאנחנו כותבים מתודות בדיקות, נקפיד להוסיף בתוך המתודות תיעוד בפורמט הבא:
  - בתחילת כל הבדיקות של מחלקות שקילות ולפני כל הבדיקות של מקרי גבול נוסיף הערות כותרת:

```
// ============ Equivalence Partitions Tests ==============
...
// =============== Boundary Values Tests ==================
...
```

  - לפני כל **test case** נוסיף הערה המתארת את המקרה המסוים הנבדק במשפט אחד
    - נמספר ונתאר תסריט של כל **test case** בהערה הזו
    - ראו דוגמה בהמשך
    - שימו לב ש-**test case** מייצג תסריט בדיקה פונקציונלית – גם אם אתם עושים בתוכו כמה **assert**-ים
- נמחק את השורה הבאה: `fail("Not yet implemented");`

- נוסיף את כל הבדיקות שתכננו, למשל דוגמא איך יכולה להיראות מתודת בדיקה (בדיקה שלכם של מכפלה וקטורית לא חייבת לפי הדוגמה, יתכן שפשוט תרצו להשוות לווקטור תוצאה מצופה, ותו לא):

```
/**
 * Test method for
 * {@link primitives.Vector#crossProduct(primitives.Vector)}.
 */
@Test
void testCrossProduct() {
   // ============ Equivalence Partitions Tests ==============
   Vector vr = v123.crossProduct(v03M2);

   // TC01: Test that length of cross-product is proper (orthogonal vectors taken
   // for simplicity)
   assertEquals(v123.length() * v03M2.length(), vr.length(), DELTA, "crossProduct() wrong result length");
   assertEquals(0, vr.dotProduct(v123), "crossProduct() result is not orthogonal to 1st operand");
   assertEquals(0, vr.dotProduct(v03M2), "crossProduct() result is not orthogonal to 2nd operand");

   // =============== Boundary Values Tests ==================
   // TC11: test zero vector from cross-product of parallel vectors
   assertThrows(IllegalArgumentException.class, () -> v123.crossProduct(vM2M4M6), //
                "crossProduct() for parallel vectors does not throw an exception");
}
```

3) נממש את הבדיקות של פעולות וקטוריות (בדומה למה שמופיע מהתוכנית הראשית של שלב 1) למתודות בדיקה המתאימות (מתודת בדיקה לכל פעולה וקטורית שנבדקת – בתוך כל מתודת בדיקה יהיו כל הבדיקות של אותה הפעולה הוקטורית) של המחלקות **VectorTests** ו-**PointTests** בהתאם (לפי המתודות הנבדקות) **תוך התאמות לסביבת בדיקות אוטומטיות של Junit**

- יש לבדוק את **כל** המתודות של **Point** ו-**Vector**, **למעט** **toString** ו-**equals** (בשניהם), ו**למעט** שני הבנאים ב-**Point**
- **תזכורת**: לא לשכוח לבדוק גם את הפעולה של חיסור וקטורים (מרות שהיא אינה ממומשת ב-**Vector**) – כי בבדיקות אנחנו "לא יודעים" שום פרטים על המימוש

4) עבור **מישור**: נוסיף בדיקה של בנאי שמקבל שלוש נקודות:

- מחלקת שקילות
  - צריך לבדוק שהנורמל מאונך לפחות לשני וקטורים שונים (בין הנקודות) ושהוא באורך=1 (תדאגו לנתונים כך שמכפלה וקטורית בין שני וקטורים כלשהם שין הנקודות תיצור וקטור באורך שונה מ-1)
- חמישה מקרי גבול\קצה (בכולם יש לוודא שנזרקת חריגה נכונה):
  - שלושה מקרים של זוגות נקודות מתלכדות (ראשונה ושניה, ראשונה ושלישית, שניה ושלישית)
  - מקרה שכל הנקודות מתלכדות
  - מקרה שכל הנקודות נמצאות על אותו ישר (אך אינן מתלכדות)

5) נוסיף למחלקות בדיקות הגופים את בדיקות המתודה **getNormal(Point point)** במודולים נפרדים עבור כל גוף, בכתיבת הבדיקות (טסטים) האלה יש לדאוג מראש ש**תמיד הנקודה בארגומנט המתודה תהיה על פני שטח הגוף הנבדק**, **חובה לבנות את הבדיקות לפני מימושים של המתודה**! כמו כן:

- בתיבת ההגשה של השלב ניתן לסטודנטים מודול **PolygonTests.java** מוכן שכולל את מתודת בדיקות הבנאי ואת מתודת בדיקות הנורמל של מצולעים – יש להעתיק את המודול לתת-חבילה **unittests.geometries** של הפרויקט שלכם **ולעיין בו על מנת ללמוד** איך לבנות מודול ומתודות של בדיקות
- בבדיקות נורמל של מישור, מצולע ומשולש [**חובה לבדוק בכולם**] – בלי קשר לדרך המימוש כפי שתפורט בהמשך – אנחנו עובדים ע"פ **TDD**! יש רק מחלקת שקילות (**EP**) אחת ואין מקרי גבול (**BVA**).
  - בבדיקה של המישור יש ליצור מישור עם בנאי שמקבל 3 נקודות (ולא ע"י בנאי שמקבל נקודה ווקטור).
  - יש לשים לב שגודל וקטור הנורמל חייב להיות שווה ל-1, ובכל הצורות השטוחות אין לדעת מראש באיזה משני הכיוונים יהיה וקטור הנורמל – הבדיקה מצליחה אם אחד האפשרויות מצליחה
- תזכורת: בבדיקות **נורמל של ספירה** גם יש רק מחלקת שקילות אחת ואין מקרי גבול
- תזכורת: בבדיקות **נורמל של גליל אין סופי** יש מחלקת שקילות אחת ומקרה קצה אחד (כאשר חיבור של הנקודה לראש הקרן של ציר הגליל מייצר זווית ישרה עם הציר – הנקודה "נמצאת מול ראש הקרן")
- עבור מי שעושה חישוב **נורמל של גליל סופי** – בבדיקות נורמל של גליל סופי יש:
  - שלוש מחלקות שקילות (על המעטפת ובתוך כל אחד משני הבסיסים)
  - ארבעה מקרי גבול (שני מרכזי הבסיסים ושני החיבורים בין הבסיסים למעטפת)

6) **עכשיו** נממש את המתודה **getNormal(Point point)** עבור כל הגופים הרלוונטיים – ע"פ הנלמד בקורס (עבור ספירה, מישור וגליל אין סופי (צינור))

- **יש להניח** שהנקודה שמתקבלת בפרמטר נמצאת על פני שטח הגוף ואין צורך לבדוק את תקינות הנקודה
- יש לממש את המתודה רק במחלקות המתאימות (לפי הארכיטקטורה)
- **במישור**, המתודה תחזיר את ערך השדה של נורמל, **אך בשלב הזה נשלים את הבנאי עם 3 נקודות** כך שיחשב את הנורמל לפי מה שלמדנו – במודל המתמטי של נורמל למשולש
- **במצולע**, המתודה כבר ממומשת מהשלב הקודם ע"י האצלה לפעולה מאחזרת של נורמל המישור המוכל
- **במשולש**, מימוש המתודה לא נדרשת – המימוש מתקבל בירושה מהמצולע
- בספירה ובגליל, המתודה תמומש לפי המודל המתמטי שלמדתם בקורס התאורטי
- **אין חובה לממש את הפעולה עבור גליל סופי – מי שיממש בצורה נכונה – יקבל בונוס של 1 נק' לציון הכללי**

נספח: תבנית בדיקת חריגה:

```
assertThrows(IllegalArgumentException.class, () -> function call, "failure text");
```

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**

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

שימו לב – אי הקפדה של ההנחיות לעיל עלולה לגרור הורדה בציון