שלב 2 – בדיקות אוטומטיות ומימוש חישובי נורמלים

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

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.

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**.

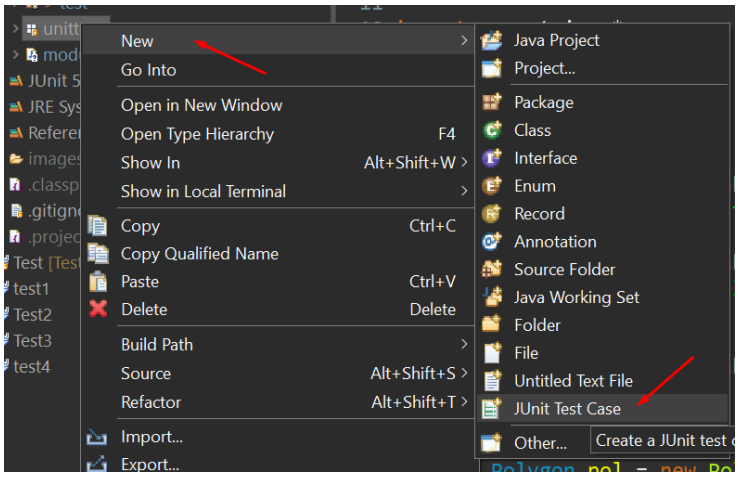
נ.ב. גרסת **JUnit** הנדרשת הנה גרסה 5 **(Jupiter)**

ההדגמה במסמך הזה מתבצעת על בסיס סביבת פיתוח **eclipse**. 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.

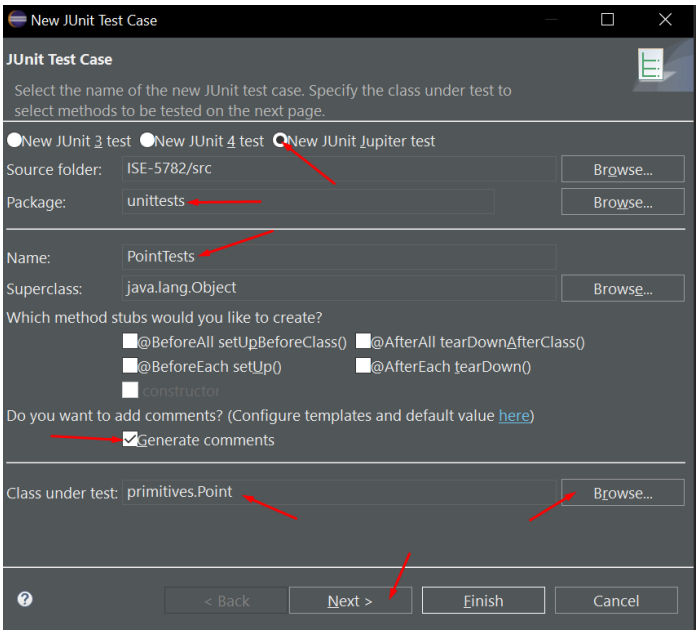
1) נוסיף לפרויקט את ספריית הבדיקות האוטומטיות **JUnit** ע"פ ההנחיות שקיבלנו בכיתה

2) נוסיף חבילת בדיקות (טסטים) **unittests**, ובה מחלקות **PointTests** ו-**VectorTests** ומחלקות בדיקה עבור כל גוף גאומטרי בהתאם (**PlaneTests**, וכו'). שים לב בשלב הזה של התרגיל עדיין **אסור** לממש את הפעולות **getNormal(Point)** במחלקות הגופים הגאומטריים. הערות כלליות לכתיבת הבדיקות:

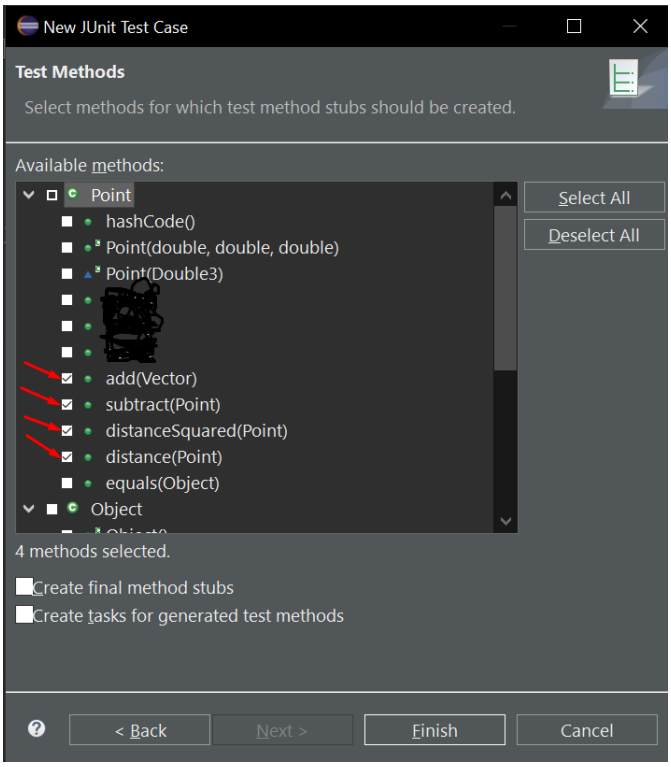
- על מנת ליצור מודול (מחלקה) לבדיקת מחלקה מסוימת מהפרויקט נעמוד עם העכבר מעל שם החבילה **unittests**, נפתח תפריט הקשר (לחיצה על כפתור ימין של העכבר) ונוסיף משם בדיקה חדשה:



- 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**:



- נעשה את הפעולות כנ״ל עבור כל מחלקת בדיקות
- בשלב הבא קודם כל נשלים את תעוד ה-**javadoc** של המחלקה והפונקציות כנדרש:

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

- כשאנחנו כותבים בדיקות בפונקציות, נקפיד להוסיף הערות כדלקמן:
  - 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
- נמחק את השורה הבאה: `fail("Not yet implemented");`
- ונוסיף את כל הבדיקות שתכננו, למשל דוגמא איך יכולה להיראות פעולת בדיקה:

```
    /**
     * Test method for {@link primitives.Vector#crossProduct(primitives.Vector)}.
     */
    @Test
    public void testCrossProduct() {
        Vector v1 = new Vector(1, 2, 3);

        // ============ Equivalence Partitions Tests ==============
        Vector v2 = new Vector(0, 3, -2);
        Vector vr = v1.crossProduct(v2);

        // TC01: Test that length of cross-product is proper (orthogonal vectors taken
        // for simplicity)
        assertEquals(v1.length() * v2.length(), vr.length(), 0.00001, "crossProduct() wrong result length");

        // TC02: Test cross-product result orthogonality to its operands
        assertTrue(isZero(vr.dotProduct(v1)), "crossProduct() result is not orthogonal to 1st operand");
```

```
        assertTrue(isZero(vr.dotProduct(v2)), "crossProduct() result is not orthogonal to 2nd operand");

        // =============== Boundary Values Tests ==================
        // TC11: test zero vector from cross-productof co-lined vectors
        Vector v3 = new Vector(-2, -4, -6);
        assertThrows(IllegalArgumentException.class, () -> v1.crossProduct(v3),
                     "crossProduct() for parallel vectors does not throw an exception");
    }
```

3) נעביר את הבדיקות (טסטים) של פעולות ווקטוריות מהתוכנית הראשית של שלב 1 לפונקציות בדיקה המתאימות (פונקציית בדיקה לכל פעולה וקטורית שנבדקת – בתוך כל פונקציית בדיקה יהיו כל הבדיקות של אותה הפעולה הוקטורית) של המחלקות **VectorTests** ו-**PointTests** בהתאם (לפי המתודות הנבדקות) **תוך התאמות לסביבת בדיקות אוטומטיות של Junit**.

4) נוסיף למחלקות בדיקות הגופים את בדיקות (טסטים) פעולת **getNormal(Point point)** במודולים נפרדים עבור כל גוף, בכתיבת הבדיקות (טסטים) האלה יש לדאוג מראש שתמיד הנקודה בארגומנט הפעולה תהיה על פני שטח הגוף הנבדק, **חובה לבנות את הבדיקות (הטסטים) קודם כתיבת המימושים של הפונקציה**! כמו כן:

- בתיבת התרגיל ניתן לסטודנטים מודול **PolygonTests.java** מוכן שכולל את פוקנציית בדיקות הבנאי ואת פונקציית בדיקות הנורמל של מצולעים, יש להעתיק את המודול לחבילה\תיקיה **unittests** של הפרויקט שלכם **ולעיין בו על מנת ללמוד** איך לבנות מודול ופונקציות של בדיקות.
- בבדיקות (טסטים) 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 וקטור).
  - 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.
- בבדיקות נורמל של ספירה גם יש רק מחלקת שקילות אחת ואין מקרי גבול
- 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 לגליל סופי – בבדיקות נורמל של גליל סופי יש שלוש מחלקות שקילות (בצד ובכל אחד משני הבסיסים) ושני מקרי גבול – במרכזי הבסיסים של הגליל. אין צורך לבדוק נקודות על קו החיבור בין בסיסים לבין החלק הגלילי של הגליל סופי מכיוון שלא ייווצרו נקודות כאלה בסופו של דבר על פני הגליל.
- עבור מישור, יש להוסיף בדיקה של בנאי שמקבל שלוש נקודות ולכלול שני מקרי קיצון:
  - הנקודה הראשונה והשנייה מתלכדות
  - הנקודות נמצאות על אותו ישר

5) **עכשיו** נממש את הפעולה **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 הפעולה רק במחלקות המתאימות (לפי הארכיטקטורה)
- **במישור** הפעולה 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** הפעולה כבר ממומשת מהשלב הקודם ע"י האצלה לפעולה מאחזרת של נורמל המישור המוכל
- **במשולש** מימוש הפעולה לא נדרשת – המימוש מתקבל בירושה מהמצולע
- בספרה ובגליל הפעולה תמומש לפי המודל המתמטי שלמדתם בקורס התאורטי (ראו גם בשקפים של המעבדה)
- **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");
```