**שלב 1 - הגדרת פרימיטיבים**

**כללים שכלים על כל השלבים של הפרויקט (כולל השלבים הנקראים "תרגיל"):**

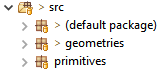
* חובה לעבוד בזוגות ע"פ עקרונות **Pair Programming** במסגרת **XP** (**Extreme Programming**) כפי שנלמד בקורס "מבוא להנדסת תכנה"
* חובה להקפיד על המוסכמות כפי שנלמד בתרגול הראשון:

1. שמות בעלי משמעות ולפי **CamelCase\ camelCase**בהתאם
2. עבור {} - שימוש ב-**Java Style**
3. הזחות, ריווח בתוך שורה
4. שורות רווח – אל לא רצף של שורות רווח

* חובה תיעוד בפורמט **JavaDoc** לכל מחלקה, כל הגדרת פעולה (אין צורך בתיעוד בפורמט הזה להעמסות – **Override**)
* בקוד המממש מודל מתמטי שאינו טריויאלי – חובה תיעוד בתוך הקוד המפרט את המודל המתמטי
* כבר מהשלב הראשון חובה לפתוח **git repo** לפרויקט ולסנכרנו (לאחר "פרסום" ראשוני) באתר **github** או **gitlab**, הפרויקט יוגדר באתר כ-**private**. לא יתקבלו נימוקים של קריסת מחשב\דיסק או כל אסון טבע אחר
* בתיבת הגשה יש להגיש בהגשה מקוונת בתיבת הטקסט את הקישור לפרויקט באתר שבחרתם כנ"ל – כולל **טאג** ההגשה

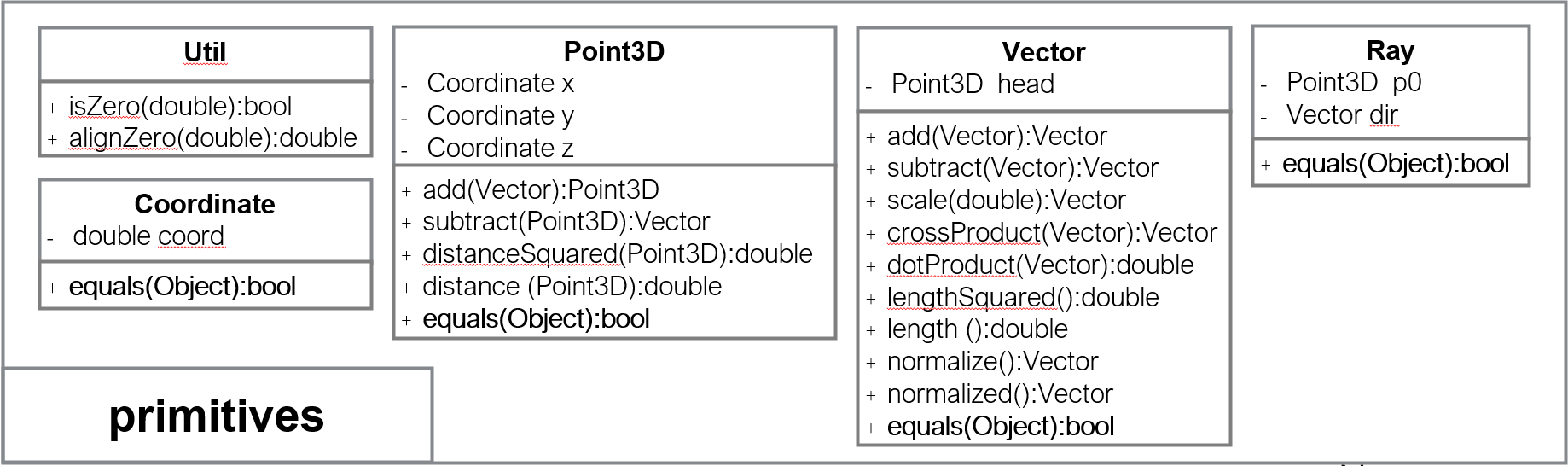
בשלב ראשון של הפרויקט נרצה להגדיר אוסף של מחלקות בעזרתם נוכל לתאר סצנה גרפית.

אבל לפני שנתחיל להוסיף מחלקות לפרויקט, נפתח שתי חבילות הנדרשות בארכיטקטורה של המערכת הנבנית: **primitives** (עבור מחלקות הייסוד ליישויות של גאומטריה), ו-**geometries** (לגופים גאומטריים). עבור כל חבילה נפתחת תת-תיקיה בשם זהה בתיקית קוד מקור של הפרויקט (**src**).

נתחיל מהוספת המודולים שקיבלתם לתרגיל:

* **Util.java** בחבילה (תיקיה) **primitives**
* **Coordinate.java** בחבילה (תיקיה) **primitives**
* **Polygon.java** בחבילה (תיקיה) **geometries**
* **Main.java** בתיקיה הראשית של קוד מקור (**src**) ללא חבילה – תראו בעץ הפרויקט ש"התווספה" חבילה\תיקיה מדומה בשם "**(default package)**" [הדוגמה לעיל מאקליפס).

נתחיל בהגדרת **חבילה של פרימיטיבים** בשם **primitives** שכוללת את המחלקות הבאות:



* **Util** (שרות – כלי עזר) - לטיפול בדיוק החישובים (ניתנת באתר **moodle** ויש להשתמש בה ללא שינויים).
* **Coordinate** (קאורדינטה) - יחידה על ציר המספרים (ניתנת באתר **moodle** ויש להשתמש בה ללא שינויים).
* **Point3D** (נקודה במרחב) - אובייקט ייסוד בגאומטריה – נקודה בעלת 3 קואורדינטות
* **Vector** (וקטור) - אובייקט ייסוד בגאומטריה בעל כיוון וגודל, מוגדר ע"י נקודת הסוף (כאשר נק' התחלה –  
   ראשית הצירים).
* **Ray** (קרן) - אובייקט ייסוד בגאומטריה – קבוצת הנקודות על ישר הנמצאות בצד אחד יחסית  
   לנקודה נתונה על הישר שנקראת ראשית הקרן. מוגדר ע"י נקודה וכיוון (**וקטור יחידה**)...

היחס בין המחלקות כדלקמן הנ"ל ע"י הכלה בלבד.

כדי לאפשר עבודה עם האוביקטים השונים נוסיף בכל מחלקה:

* **constructors** (עם פרמטרים/"העתקה"). לא יוגדרו בנאיי ברירת מחדל.
* תוגדרנה פעולות אחזור (**get**) לשדות איפה שלא נאמר במפורש שהן לא תוגדרנה
* לא תוגדרנה פעולות עדכון (**set**)
* העמסת\דריסת פעולה **еquals** ע"פ הכללים שנלמדו ותוך שנירה על העיקרון **DRY** (חובה להתבסס על **equals** של האוביקטים המוכלים). תבנית ההעמסה כדלקמן (כולל דוגמת הגדרת מחלקה):

**package** primitives;

/\*\*

\* Class ***ClassName*** is the basic class representing a … of Euclidean geometry in Cartesian

\* 3-Dimensional coordinate system.

\* **@author** Student1 and Student2

\*/

**public** **class** ***ClassName*** {

…

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj) **return** **true**;

**if** (obj == **null**) **return** **false**;

**if** (!(obj **instanceof** ***ClassName***)) **return** **false**;

***ClassName*** other = (***ClassName***)obj;

**return** this.field1.equals(other.field1) && this.field2.equals(other.field2) …;

}

}

* העמסת\דריסת פעולה **toString** על מנת לאפשר הדפסה של האובייקט – בצורה תמציתית אך נוחה לשימוש (שימו לב בהדפסת קואורדינטה, נקודה, ווקטור – יהיו לנו אובייקטים עם מספר נקודות, בכל נקודה יש שלוש קואורדינטות)! חובה להתבסס על מה שכבר קיים (לשמור על העיקרון **DRY**)
* חובה שהבנאי של **וקטור** יזרוק חריגה במקרה של וקטור אפס, אסור להשתמש בחריגת בסיס **Exception** אלא למצוא את החריגה המתאימה בין החריגות המוגדרות בספריות **java**.
* חובה שהבנאי של **קרן** יבטיח שווקטור הכיוון מנורמל.

בנוסף לכל מחלקה נוסיף פעולות שמאפשרות עבודה עם אובייקטים בטיפוס זה (אף פעולה למשנה את האובייקט שלה חוץ ממקרה אחד המצוין במפורש בהמשך):

**שירות: המחלקה ניתנת באתר moodle**

**קאורדינטיה: המחלקה ניתנת באתר moodle**

**נקודה במרחב:**

* השדות של הקואורדינטות מסוג **double**, ללא הרשאה (= הרשאת חבילה)
  + לשיקול הסטודנטים ניתן גם להגדיר את השדות ב-**private** ולהוסיף פעולה מאחזרת שתחזיר את הערך של קואורדינטה מטיפוס **double** כפי שנלמד בקורס התאורטי

**נ.ב.** ההמלצה הראשונה לעיל ניתנה במבט קדימה לביצועים בפרויקט (אמנם זה לא "אג'ייל-י" אבל יחסוך זמן בהמשך

* בנאים המקבלים: א) שלוש קואורדינטות, ב) שלושה מספרים מסוג **double**

**נ.ב.** אסור ליצור בנאי העתקה

* קבוע סטטי **ZERO** המכיל נקודה (0,0,0)
* במחלקה **לא יהיו** פעולות מאחזרות (**get**) בכלל
* **subtract** – חיסור וקטורי – מקבל נקודה שניה בפרמטר, מחזיר וקטור מהנקודה השניה לנקודה שעליה מתבצעת הפעולה
* **add** – הוספת וקטור לנקודה – מחזיר נקודה חדשה
* **distanceSquared** – המחרק בין שתי נקודות בריבוע
* **distance** – מרחק בין 2 נקודות (יש להשתמש בפעולה הקודמת)

**נ.ב.** **אסור** להשתמש בפונקציה **Math.pow()** על מנת לחשב ערך בריבוע. הפונקציה הנ"ל מחשבת גם חזקות עם מעריכים ממשיים בעזרת חישובים מקורבים ע"י טורים מתמטיים. לא כדאי להשתמש - הולכים לעשות המון חישובים. תכפילו את הערך בעצמו!

**נ.ב.2** **אסור** ליצור פעולות נוספות אלא אם השתכנעתם שאתם צריכים אותן ויש להן משמעות גאומטרית או אלגברית!

**וקטור** (בכל הפעולות יש להשתמש בשיטה של אלגברה לינארית ולא בשיטה טריגונומטרית):

* בנאים המקבלים: א) שלוש קואורדינטות, ב) fשלושה מספרים מסוג **double**, ג) נקודה

**נ.ב.** אסור ליצור בנאי העתקה

* כל הבנאים (חוץ מהאחרון) "יזרקו" חריגה מסוג **IllegalArgumentException** אם טקסט מתאים במקרה של "וקטור 0". חובה להשתמש ב-**equals** של נקודה עבור הבדיקה
* **subtract/add** – חיבור/חיסור ווקטורי (מחזירים וקטור חדש)
* **scale** – מכפלת ווקטור במספר - סקלר (מחזירה וקטור חדש)
* **dotProduct** – מכפלה סקלארית (**dot-product**) – יש להשתמש בנוסחה של אלגברה לינארית ולא טריגונומטרית  
  *[בבדיקות יש לבדוק וקטורים עם אותו כיוון, עם זוית חדה ביניהם, אורתוגונלים, עם זוית כהה, עם כיוונים הפוכים (על אותו ישר)]*
* **crossProduct** – מכפלה ווקטורית – מחזירה וקטור חדש שניצב לשני הוקטורים הקיימים (**cross-product**), יש להשתמש בנוסחה של אלגברה לינארית ולא טריגונומטרית, *[הבדיקות כנ"ל]*
* **lengthSquared** – חישוב אורך הווקטור בריבוע
* **length** – חישוב אורך הווקטור (יש להשתמש בפעולה הקודמת)
* **normalize** – פעולת נירמול הווקטור שתשנה את הוקטור בעצמו (הפעולה היחידה שמשנה את האוביקט שעליו זומנה), הפעולה גם תחזיר את הוקטור לצורך שרשור הפעולות במקרה הצורך (**return this;**)
* **normalized** – פעולת נרמול שמחזירה וקטור חדש מנורמל באותו כיוון כמו הוקטור המקורי (יש להשתמש בפעולה הקודמת)

**קרן** ("חצי-ישר – כל הנקודות על הישר הנמצאות בצד אחד מהנקודה הנתונה על נהישר הנקראת ראשית\ראש\תחילת הקרן):

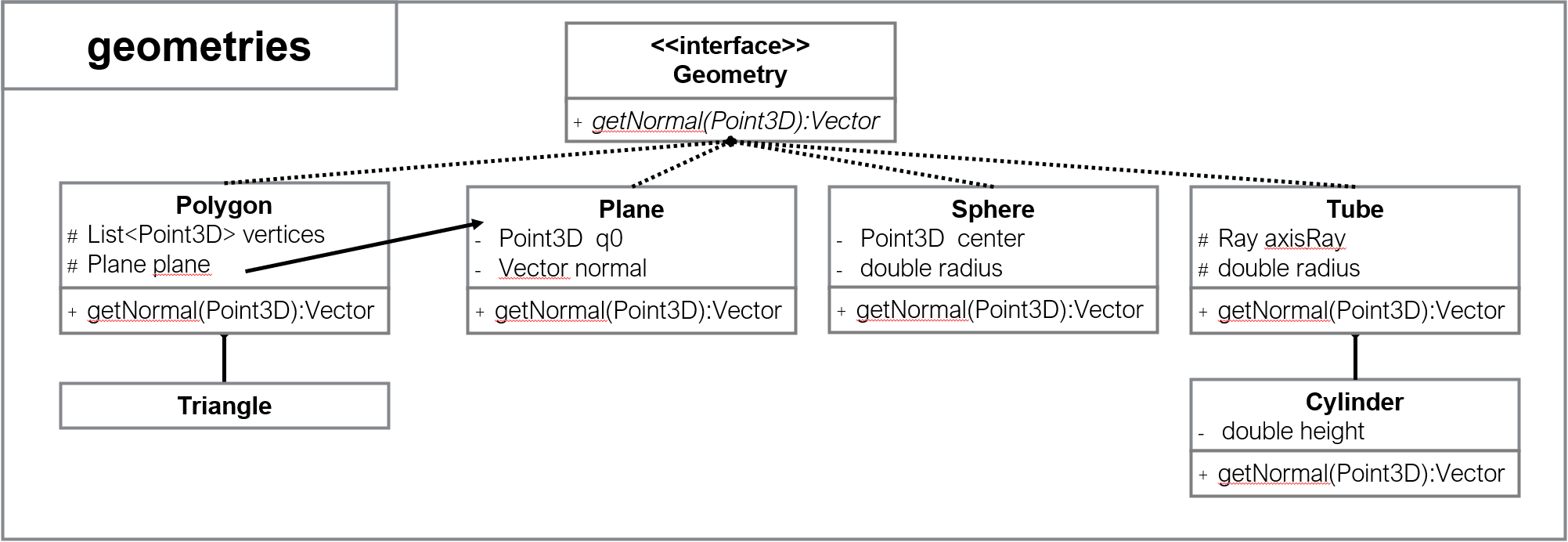
* בנאי המקבל: נקודה ווקטור (וקטור שמתקבל לבנאי לא חייב להיות מנורמל, אך וקטור הנשמר בקרן חייב להיות מנורמל)

**נ.ב.** אסור ליצור בנאי העתקה

בעזרת חבילת הפרימיטיבים שהגדרנו נוכל עכשיו להגדיר אוסף של מחלקות בעזרתם נוכל לתאר צורות וגופים הנדסיים (גיאומטריים).

בדומה למה שעשינו קודם נגדיר **חבילת הצורות** בשם **geometries** שמכילה אוסף של מחלקות לתיאור צורות וגופים גאומטריים. כדי לאפשר לבנות סצנה מאוסף של גופים שונים נבנה את החבילה בצורה היררכית.

נגדיר **ממשק** (**interface**) בשם **Geometry** עבור גוף גאומטרי כלשהו ואחר כך נגדיר מחלקות עבור הגופים השונים לפי הארכיטקטורה הבאה:



לממשק נוסיף פונקציה בשם **getNormal** המקבלת פרמטר אחד מסוג נקודה [על פני הגוף הגאומטרי] ומחזירה ווקטור הנורמל (האנך) לגוף בנקודה זו.

נבחר במספר גופים גאומטריים למימוש בפרויקט (ברור שניתן להרחיב את הפרויקט לגופים רבים נוספים).

לכל אחת מהמחלקות של הגופים האלה עליך להוסיף:

* בנאים (**אסור** להגדיר בנאיי ברירת מחדל ובנאיי העתקה!]
* פעולות אחזור (**getters**) בהתאם (**אסור** להוסיף פעולות עדכון(**setters**))
* העמסות של **toString** (**אסור** להעמיס **equals**)

**צורות שטוחות:**

* מישור (נקודה במרחב ווקטור אנך). [**Plane** או **Plain** לפי הניב של אנגלית המועדף עליכם]
  + שני בנאים – אחד עם 3 נקודות ואחד עם נקודה ווקטור הנורמל
  + בנאי עם 3 נקודות אמור לחשב את הנורמל לפי מה שנלמד על נורמל למשולש – בשלב הזה יישמר ערך **null** משדה הנורמל (המימוש המלא יתבצע בשלב הבא), כמו כן הבנאי ישמור את אחת הנקודות כנקודת הייחוס של המישור
* מחלקה של מצולע. [**Polygon**]. **המחלקה מצורפת לתרגיל במלואה לפי השלב הזה** (ראו בסוף הקובץ).
  + המחלקה תכיל אוסף ממוספר של קודקודי המצולע:

**List<Point3D> vertices = new ArrayList<Point3D>();**

* + כמו כן החלקה תכיל את [הפניה ל-] המישור שהמצולע נמצא בו
  + הבנאי יקבל נקודות הקודקוד של פוליגון ע"פ הסדר שלהם מבחינת חיבור ביניהם ע"י צלעות המצולע ויבדוק האם הרשימה של הקודקודים של המצולע תקינה (מסודרת לפי חיבורי הצלעות, אין קודקודים על אותו ישר, המצולע קמור) [הקוד מבוסס על גרסה 9 ומעלה של **Java**) – ראו בקוד המצורף
    - השימוש בפרמטר **vertices** הנ"ל בתוך הבנאי הוא כמו במערך של נקודות – **vertices[i]**
  + הבנאי יוודא שיש לפחות 3 קודקודים (אחרת תיזרק חריגה מסוג **IllegalArgumentException**)
  + הבנאי יצור מישור ע"י שלושת הנקודות הראשונות
* משולש (מצולע בעל קודקודים). [**Triangle**]
  + יירש מהמצולע
  + הבנאי יקבל שלוש נקודות ויפעיל את בנאי האב בהתאם

**גופים תלת מימדיים**:

* כדור (נקודה ורדיוס) [**Sphere**]
* צינור (רדיוס וקרן) [**Tube**]
* גליל (גובה) [**Cylinder**]

כל המימושים לפעולה **getNormal** של הממשק **Geometry** בשלב הזה יחזירו **null**, למעט הפוליגון (המצולע) שיחזיר את הנורמל של המישור המוכל בו (ז"א האצלת פעולה).

**תוכנית ראשית שבודקת את נכונות הפעולות מצורפת לתרגיל**. יש לוודא שהבדיקות מסתיימות בהצלחה.

ציון התרגיל יינתן לפי אחוז הכיסוי (הביצוע) של משימת התרגיל – כולל בדיקות כל הפעולות, כמו כן על ההקפדה על הכללים.

**שימו לב** – אי שמירה של ההנחיות לעיל עלולה לגרור הורדה בציון.