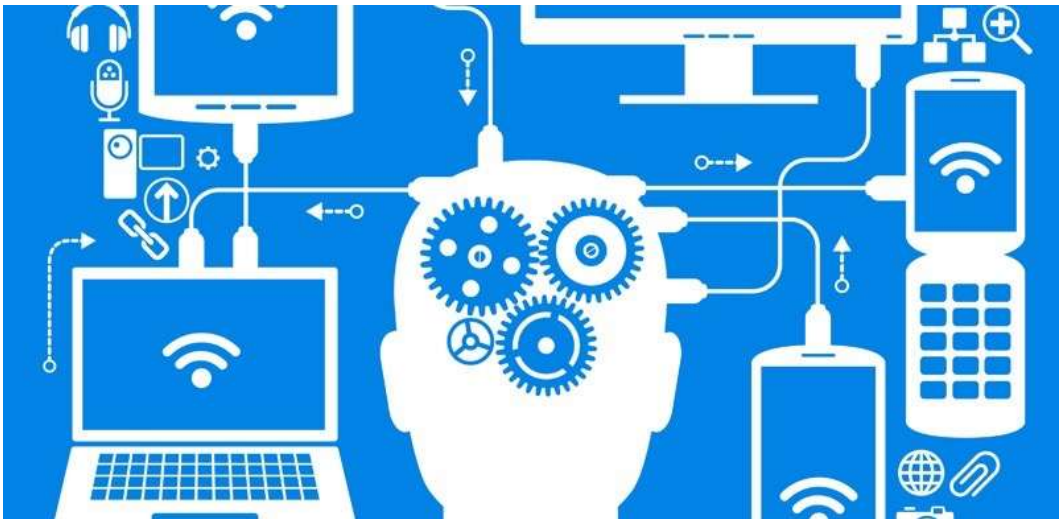
Introduction to Software Engineering

JERUSALEM COLLEGE OF TECHNOLOGY
LEV ACADEMIC CENTER



Recitations

- *Week 0: Home assignment 0 (not graded): Introduction to OOP with Java.*
- **Week 1: Home assignment 1**: Implementation of primitives with operations, geometries
- *Week 2: Home assignment 2: Implementation of geometries primitives' unit testing for primitives (JUnit). Implementation of normal calculation and their unit testing through geometries.*
- *Week 3: Home assignment 3: Implementation of ray-geometry intersections and their unit testing through geometries.*
- *Week 4: Home assignment 4: Implementation of camera class, rays through view plane construction and unit testing of camera*
- *Week 5: Home assignment 5: Implementation of ambient light, scene, render and image-writer classes with their appropriate test units*
- *Week 6-7: Home assignment 6: Adding material support. Implementation of Phong model, with light emission, directional, point and spot lighting, multiple light sources*
- *Week 8: Home assignment 7: Implementation of shadow rays, reflection and refraction*
- *Week 9: Mini-project 1: Implementation of a picture improvement algorithm*
- *Week 10-11: : Mini-project 2: implementation of a performance improvement algorithm*
- *Week 12-13: Mini-projects presentation*

Home Assignment 1

Implementation of geometric primitives with operations,
Testing the operations in **main** program,
Defining geometric bodies.

תרגיל (שלב) 1

מימוש פרימיטיבים גיאומטריים עם פעולות וקטוריות, השוואה והצגה

בדיקת הפעולות בעזרת תוכנית ראשית

הגדרת גופים גיאומטריים

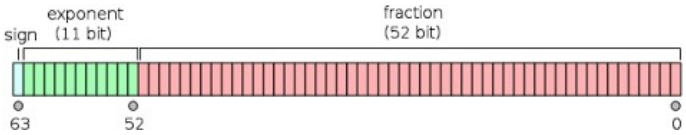
Taking Constraints in Account

Utility Class

```
public class Util {
   // It is binary, equivalent to ~1/1,000,000 in decimal (6 digits)
   private static final int ACCURACY = -40;

   // double store format: seee eeee eeee (1.)mmmm … mmmm
   // 1 bit sign, 11 bits exponent, 53 bits (52 stored) normalized mantissa
   private static int getExp(double num) {
      return (int)((Double.doubleToRawLongBits(num) >> 52) & 0x7FFL) - 1023;
   }

   public static boolean isZero(double number) {
      return getExp(number) < ACCURACY;
   }
   public static double alignZero(double number) {
      return isZero(number) ? 0.0 : number;
   }
   ... and more ...
}
```



המחלקה ניתנת לסטודנטים מוכנה ובדוקה.

שקף לתזכורת\ריענון בלבד – לא להתעכב עליו! הסטודנטים מופנים להרצאה המתאימה.

Taking Constraints in Account

```
public record Double3 (double d1, double d2, double d3) {
   public static final Double3 ZERO = new Double3(0, 0, 0);
   public static final Double3 ONE = new Double3(1, 1, 1);

   public Double3(double value) { this(value, value, value); }

   @Override
   public boolean equals(Object obj) {
      if (this == obj) return true;
      return obj instanceof Double3 other
             && Util.isZero(d1 - other.d1) &&
             && Util.isZero(d2 - other.d2)
             && Util.isZero(d3 - other.d3);
}

   @Override
   public int hashCode() { return (int) Math.round(d1 + d2 + d3); }

   @Override
   public String toString() { return "(" + d1  + "," + d2  + "," + d3  + ")" ; }
}
```

Double3
ZERO:Double3 = (0,0,0) + d1,d2,d3:double
+ equals(Object):boolean + toString():String + add(Double3):Double3 + subtract(Double3):Double3 + product(Double3):Double3 + scale(double):Double3 + reduce(double):Double3

המחלקה ניתנת לסטודנטים מוכנה ובדוקה.

שימו לב: השדה איננו פרטי - package-friendly – הוא נגיש לכל החבילה של הפרימיטיבים – בשביל אילוץ היעילות! לכן אין לכתוב גטר בשביל השדה הזה!

ועוד – אובייקטים של המחלקה לא ניתנים לשינוי - **immutable**, לכן לא צריך ליצור העתקים של קוארואדינטות בשום מקום! הדבר נעשה על ידי שימוש ב-record כטיפוס המחלקה.

שימוש ב-record מספק גם בנאי מרומז שמקבל פרמטרים כנגד השדות שמוגדרות בסוגריים בכותרת המחלקה.

בנאי נוסף שרואים בשקף יהיה שימושי בשלבים הבאים של הפרויקט.

ועוד דבר אחד –במחלקה הזו שניתנת לכם מוכנה יש עוד מתודות (חיסור נקודות, נוקדה + וקטור) שבגדול צריך לבדוק אותן ב-main, אך מכיוון שאתם מקבלים אותה מוכנה ובדוקה – לא נעשה את זה

תבנית של equals – חובה!

והערה נוספת לגבי המימוש של מתודה equals: שימו לב על החלק הראשון שב-return הארוך – הוא גם בודק את הטיפוס של האובייקט להשוואה, וגם מגדיר משתנה מהטיפוס המתאים תוך המרה אוטומטית של הפרמטר לטיפוס המתאים. ניתן להשתמש במשתנה הזה כבר בהמשך אותו הביטוי – כפי שאתם רואים בהמשך הביטוי של ה-return הזה.

יש כאן גם שני קבועים לשימוש בשלב 1 (ZERO) ובשלבים הבאים (ONE)

Code Reuse by Delegation

```
public class Point {
   final Double3 xyz;
   ...
   @Override
   public boolean equals(Object obj) {
      if (this == obj) return true;
      return obj instanceof Point other && xyz.equals(other.xyz);
}

   @Override
   public String toString() { return "" + xyz; }
}
```

```
public class Vector extends Point {
   ...
   @Override
   public boolean equals(Object obj) {
      if (this == obj) return true;
      return obj instanceof Vector other && super.equals(other);
}

   @Override
   public String toString() { return "->" + super.toString(); }
}
```

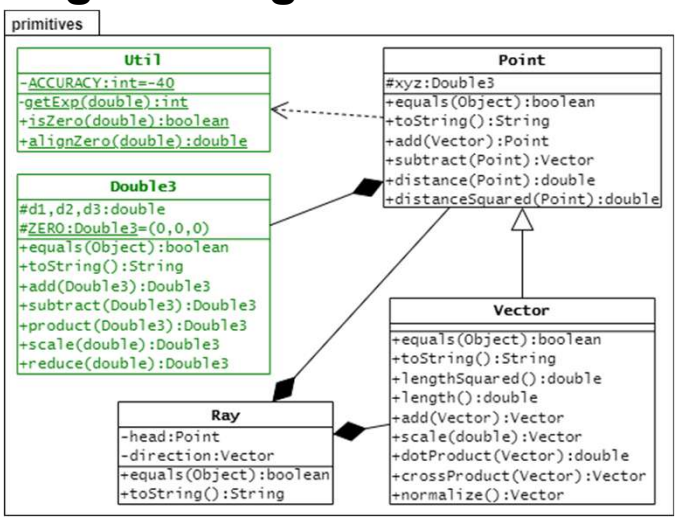
תבנית חובה עבור שלושת הפונקציות האלה!

מניעת **needless repetition**, התבססות על עיקרון **DRY**.
שורות ראשונות חוזרות על עצמן –השורות האלה הן משהוא שאי אפשר להוציא לפוקנציה נפרדת ואין בררה – חייבים לכתוב למרות שזה לא נראה יפה. עם כל העקרונות – יש גם אילוצים.

החלקים של הבדיקה :

1. אם אובייקט להשוואה הוא בעצם אותו אוביקט (אותה הפניה) – ברור שכל אובייקט שוה לעצמו
2. אם אובייקט להשוואה איננו מופע של הסוג של האובייקט שלנו (זה כולל התייחסות למצב ש-obj לא קיים – שווה ל-null, במקרה כזה הוא לא יהיה "סוג של" מתאים) – ברור שהם לא שוים
3. שימו לב על יכולת להגדיר משתנה תוך כדי המרה ("כלפי מטה" – לפי חוקי ירושה) לטיפוס המתאים - **בתוך פעולת instanceof שבתוך הבדיקה** (תקף רק לבלוק של התנאי) – חוסך שורה של הגדרת משתנה
4. הפעולות מתבצעות לבסוף לא ע"י פעולה ישירה בתוכן השדות אלא ע"י האצלה לטיפוסים שלהם - דרך הפעלת אותה הפונקציה על האובייקטים של השדות (אלא אם מדובר בסוגים פרימיטיביים) – בכל שלושת הפונקציות

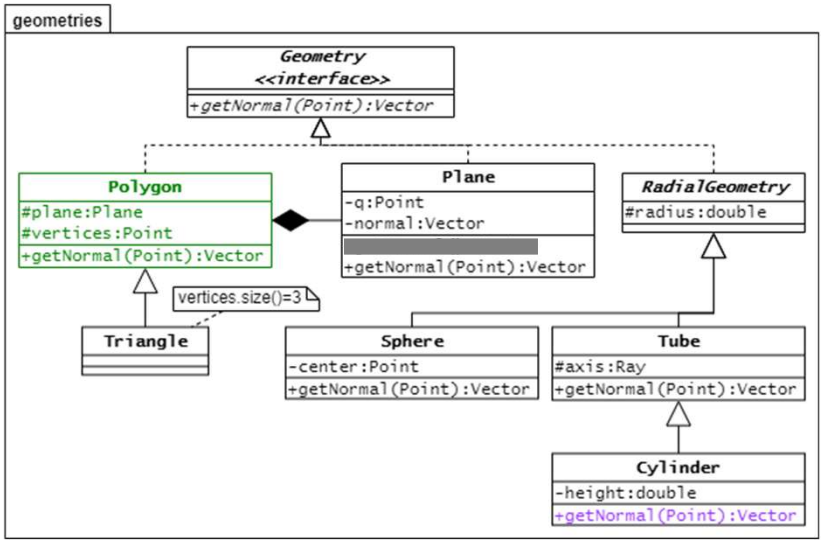
Packages Design



- All classes are immutable
- Each operation returns a new object
- Zero Vector throws exception **IllegalArgumentException**
- DRY principle => software reuse
- Ray's ctor normalizes its **direction** vector

- כל האובייקטים של פרימיטיבים לא ניתנים לשינוי (**immutable**) על מנת להבטיח שבירות נמוכה (low fragility) בהמשך הפיתוח. לכן כל הדות מוגדרים **final**, ופונקציות של כל הפעולות על האובייקט מייצרות ומחזירות אובייקט חדש שהוא תוצאת הפעולה בהתאם
- לא דורסים (מממשים) את פונקצית חיסור הווקטורים – כי היא כבר מממומשת במחלקת האב – Point
- בהמשך נידרש לרוב לדעת את ערך אורך הוקטור בריבוע – אז למה לחשב את השורש ואחר כך אחרים יעלו בריבוע? ניתן פונקצייה של ריבוע האורך
- לא לשכוח על הביצועיים – מימוש מלא של הפונקצית אורך בריבוע, והפונקציה של אורך תשתמש בה! (מניעת כפל קוד תוך שמירה על הביצועים)
- **בקורס הזה אסור ליצור וקטור אפס** (0,0,0), בבנאי של ווקטור חובה לבדוק האם נוצר ווקטור אפס (חובה להשתמש בהשוואה של שליישת [Double3] הקואורדינטות xyz לשלישיית אפס - חפשו את הקבוע המתאים במחלקה Double3!)
- בכל מקרה – כל השוואת של מספרים ל-0 מתבצעות בעזרת פונקציות Util : פונקציה isZero במקום ==, ופונקצייה alignZero כאשר נדרשת השוואה יחסית - >\>=\<\<=
- בכל המימושים חובה לחשוב ולהימנע משימוש בקוד שכבר קיים (אנטי-תבנית "העתק\הדבק") – שמירה על עיקרון DRY. השתמשו במידת האפשר בפעולות שכבר קיימות ב-Double3!
- בנקודה ובורטור חובה ליצור שני הבנאים הבאים : (1) מקבל שלושה מספרים (עבור הקואורדינטות), והשני - אובייקט Double3
- בקרן חובה לנרמל את ווקטור הכיוון לפני השמירה – אין הבטחה שנקבל אותו בפרמטר כאשר הוא כבר מנורמל

Packages Design



- All classes are immutable
- All getNormal overrides return null but Plane's one returns **normal**
- Plane ctor's: 1) by 3 points, 2) by point and vector

- גם כאן כל האובייקטים של הגופים הגיאומטריים לא ניתנים לשינוי (**immutable**) על מנת להבטיח שבירות נמוכה (low fragility) בהמשך הפיתוח. לכן כל השדות מוגדרים **final**, ומתודות של כל הפעולות על האובייקט מייצרות ומחזירות אובייקט חדש שהוא תוצאת הפעולה בהתאם
- **<u>אין</u>** פעולות השוואה ואין סטרים וגטרים לגופים הגיאומטריים
- אנחנו לא דורשים לדרוס את מתודת תיאור toString בגופים, אך זה יועיל לכם ולכן מומלץ מאד לדרוס אותה
- כל דריסות המתודה getNormal "ריקות" בשלב הזה (ללא מימוש אמיתי) ופשוט מחזירות **null**, למעט במישור
- **במישור**, דריסת המתודה getNormal מחזירה את ערך השדה normal
- **במישור**, חובה לעשות **שני בנאים** – אחד שמקבל שלוש נקודות (לא חייבים לממש את החישוב הנדרש, אפשר להשאיר את הבנאי הזה ריק), ועוד אחד שמקבל נקודה ווקטור נורמל (שנשמרים בשדות). לא לשכוח לנרמל את הווקטור – לא מובטח שנקבל אותו בפרמטר כשהוא כבר מנורמל.

Requirements

- Working in couples only
- Pair Programming – working together:
  - One student with the fingers on the keyboard
  - Another looking, commenting, fixing
  - Switch positions 50% / 50%
- Using GIT is mandatory
  - Project and repository name: ISE5785_XXXX_YYYY
    - XXXX and YYYY – 4 last digits of students' id's
- Using github.com is mandatory
  - **Private** repository
  - Add your teacher as collaborator
- Submit by creating a commit tag PR01 and use the URL of the tag in github.com for submission

דרישות:

- עבודה בזוגות בלבד
- שיטת עבודה – תיכנות זוגות (pair programming) – תמיד עובדים ביחד (פיזית או בזום) על מחשב של אחד השותפים:
  * סטודנט אחד כותב את הקוד בפועל ("אצבעותיו על המקלדת")
  * השני צופה כל הזמן – מעיר, מאיר, ומתקן
  * מידי פעם מחליפים את התפקידים – בסופו של דבר כל אחד צריך לכתוב באצבעותיו כמחצית מהקוד
  * החלפת תפקיד מלווה ב-commit ו-push (ראו בהמשך לגבי גיט), מעבר לעבודה במחשב של השותף השני שיתחיל מ-pull, כמובן
  * **בכל מקרה אין לעבוד לבד** (ולהשאיר את השותף מחוץ לתמונה)
- חובה להגדיר מאגר גיט מקומי אצל שני השותפים – שם המאגר כנ"ל בשקף
- חובה ליצור ולחבר את המאגר המקומי של שני השותפים למאגר בענן באתר של github.com:
  * שם המאגר זהה לשם המאגר המקומי כנ"ל
  * המאגר מוגדר כ"**פרטי**"
  * חובה להוסיף את המרצה כשותף (collaborator) במאגר הזה
- הגשת השלב באתר הקורס (ב-moodle) ע"י יצירת תג ב-commit האחרון להגשה, דחיפתו למאגר בענן והעתקת הקישור לתג בענן לתיבת ההגשה של השלב

Coding standards

- Java style only:

```
if (...) {
   ...
} else {
   ...
}
```

- Naming – PascalCase/camelCase
  - All types – PascalCase (MyClass)
  - All variables/fields/parameters – camelCase (myVariable)
  - All constants UPPER_CASE
- Correct indentation is mandatory
- Correct spacing is mandatory (in-line and between lines)

הקפדה על מסוכמות הקוד של ג'אוה – חובה!

- **שימוש ב-java-style** מבחינת מיקום הצמדיים { } – כנ"ל בדוגמה!
- **מוסכמות שמות** כנ"ל בשקף: טיפוסים (מחלקות) ע"י PascalCase, שדות, פרמטרים, משתנים מקומיים, ספונקציות – CamelCase, קבועים – אותיות גדולות עם "_"
- **חובה** - הזחות תקינות, ריווח תקין בתוך שורה
- **שורות רווח** איפה שצריך, אסור בהחלט להוסיף רצף של כמה שורות רווח – שורת רווח אחת בלבד!

נ.ב. פרמוט אוטומטי שפוטר את רוב הבעיות:

ב-eclipse: Ctrl-Shift-F

ב-IntelliJ IDEA: Ctrl-Alt-L

Documentation

- Each class and each member (method, field) must have JavaDoc style comment
  - See example in the next slide
- This documentation is **EXTERNAL**, that is it describes what is it and what is done but does NOT describe how is it done
- The documentation must be written after declaration but before you write the implementation!
- **No** JavaDoc on @Override methods
- Comments in English only!

הקפדה על תיעוד – חובה!

- **תיעוד בפורמט javadoc**: לפני כל מחלקה, כל שדה או מתודה (למעט דריסת מתודה)
- התיועד הזה הינו **תיעוד חיצוני** – זאת אומרת אסור לו לתאר או להיות תלוי במימוש של הפונקציה, התיעוד מתאר מה המטרה של הפונקציה ומה היא עושה (**לא** איך היא עושה!), איך להשתמש בה, ומה היא מחזירה (וזורקת – אם יש זריקת חריגות)
- חובה לכתוב את התיעוד כאשר אתם כותבים את הכותרת שלה ולפני המימוש שלה – זה יעזור לכם לכתוב את התיעוד בצורה נכונה ובלתי תלויה במימוש!
- אין לכתוב תיעוד בפונקציות נדרסות (@Override), למשל equals, toString
- בפונקציות המוצהרות בממשק (interface) או במחלקה אבסטרקטית – התיעוד יהיה רק לפני ההצרת המתודה בממשק ובמחלקה האבסטרקטית! וכמובן הוא יהיה "חיצוני" – כנ"ל, ובלתי תלוי במי מממש את הפונקציות
- **אין** להוסיף תיעוד לפונקמיות המממשות פונקציית ממשק או פונקציה אבסטרקטית שיורשים אותה
- **חובה** שכל התיעוד יהיה **באנגלית** בלבד

אתם מקבלים כמה מודולים בשלב הזה (Main, Util ,Double3, Polygon) – חובה לעיין בהם וללמוד מהם – גם על איך כותבים תיעוד!

Documentation Example

```
/**
 * This class will serve all primitive classes based on three numbers
 * @author Dan Zilberstein
 */
class Double3 {
   ...
   /**
    * Sum two floating point triads into a new triad where each couple of numbers
    * is summarized
    *
    * @param rhs right hand side operand for addition
    * @return result of add
    */
   Double3 add(Double3 rhs) {
      ...
```