Lombok:

ספרייה שמגנרטת קוד למחלקות כדי לקצר את האורך שלהן, לשפר את הקריאות שלהן ולחסוך מהמפתח **לממש ולתחזק** פונקציות כלליות כמו getters ו setters
הספרייה עובדת עם אנוטציות.

אנוטציות:

- רוב המחלקות ב java עובדות לפי ה Java Bean pattern, שבו השדות הם פרטיים, המחלקה מכילה פונקציות getters ו setters כדי לגשת לשדות של המחלקה, ובמחלקה יש בנאי ריק.
- getter & setter:
  - @Getter: אנוטציה שמגנרטת למחלקה פונקציות getter לשדות שלו.
    כאשר האנוטציה היא על המחלקה, זה יוצר פונקציות getter **לכל השדות** במחלקה.
    כאשר האנוטציה היא על שדה ספציפי, זה יוצר פונקצית getter לאותו שדה.
    - @Getter(lazy= true): מאפשר להגדיר lazy getter. פונקציה שתחשב את הערך של השדה רק כשצריך להשתמש בו, ותשמור את הערך ב cache כדי למנוע חישוב חוזר ומיותר.
    - @Accessors(fluent=true): אנוטציה שמאפשרת לפונקציות setXXX ו getXXX של שדה להיות השם של השדה, בלי התחילת set ו get
    - ניתן להוסיף לאנוטציוות ערך AccessLevel.XXX כדי להגדיר את ההרשאות לגישה לפונקציות שרוצים.
  - @Setter: אנוטציה שמגנרטת למחלקה פונקציות Setter לשדות שלו.
    כאשר האנוטציה היא על המחלקה, זה יוצר פונקציות Setter **לכל השדות** במחלקה.
    כאשר האנוטציה היא על שדה ספציפי, זה יוצר פונקצית Setter לאותו שדה.

- constructors:
  - @NoArgsConstructor: יוצר בנאי ריק למחלקה.
  - @AllArgsConstractor: יוצר בנאי עם כל השדות של מחלקה.
  - @RequiredArgsConstructor: יוצר בנאי לשדות שחובה להכניס להם ערך.

- פונקציות נפוצות במחלקה:
  - @ToString: אנוטציה על מחלקה. מייצר פונקצית toString גנרית שמכילה את כל השדות במחלקה.
    - @ToString(exclude=[field1,field2..]): ניתן להוריד מהפונקציה toString ערכים שלא רוצים שיכללו.
  - @EqualsAndHashCode: אנוטציה על מחלקה. מייצר פונקציות Equals ו hashCode לפי ההגדרה כאן https://www.artima.com/lejava/articles/equality.html/
    - @EqualsAndHashCode(of=[field1,field2..]): ניתן לכלול בפונקציות hashCode ו Equals רק את השדות שרוצים

- כללי:

- @NonNull: אנוטציה על שדה. מכריח את הקומפיילר לבדוק שהערך שנכנס לשדה הזה הוא לא null, זורק שגיאת NullPointerException במידת הצורך. גם בבנאי וגם בפונקצית setter של השדה.
- @FieldDefaults(makeFinal = true, level = AccessLevel.PRIVATE): אנוטציה שמאפשרת להגדיר התנהגות לכל השדות במחלקה, למשל באנטוציה @Value ע״י האנוטציה הזאת מגדירים את כל השדות פרטיים וfinal

- אנוטציות מיוחדות: (אנוטציות שנוצרו למקרים נפוצים ומאגדים תחתם מספר אנוטציות לשיפור הקריאות)
  - @Data: אנוטציה על מחלקה. מגדירה את המחלקה כdata model, ובכך מוסיפה לו את האנוטציות הבאות:
    - @Getter and @Setters
    - @ToString
    - @EqualsAndHashCode
    - @RequiredArgsConstructor
  - @Value: אנוטציה על מחלקה. מגדירה את המחלקה כ immutable שכל השדות בה הם פרטיים, סופיים וניתן לגשת אליהם רק על ידי פונקציות getter. מוסיפה את האנוטציות הבאות:
    - @Getter
    - @AllArgsconstructor
    - @EqualsAndHashCode
    - @ToStrirng
    - @FieldDefaults(makeFinal = true, level = AccessLevel.PRIVATE)
  - ניתן להגדיר לאנוטציות @Value ו @Data ערך ("staticConstructor="of"), שמגדיר בנאי סטטי בשם of. משתמשים בזה כשיש ערך גנרי.
    דוגמא: Foo.of(5) vs new Foo<Integer>(5)

- Builder pattern:
  - @Buider: מגדיר את המחלקה כתומכת ב builder pattern.
    דוגמא לבניית ערך במקרה כזה:

```
ApiClientConfiguration config =
    ApiClientConfiguration.builder()
        .host("api.server.com")
        .port(443)
        .useHttps(true)
        .connectTimeout(15_000L)
        .readTimeout(5_000L)
        .username("myusername")
        .password("secret")
    .build();
```

- זריקת שגיאות:
  - @SneakyThrows: אנוטציה על מטודה. במקרה בו רוצים לזרוק את הexception שמקבלים, האונטציה עוטפת את השגיאה ב כRunTimeException

וזורקת אותה בעצמה.

- שיחרור משאבים:
  - @Cleanup: אנוטציה על הגדרת משתנה מקומי. עבור אובייקטים שמממשים את הממשק AutoCloseable, הספרייה משחררת את המשאבים של המשתנה בסוף הסקופ שבה הוא הוגדר.(קוראת לפונקציית close שלו).

- הגדרת Logger למחלקה:
  - @Log4j2: אנוטציה על המחלקה. מגדיר logger למחלקה. ניגשים לפונקציות על ידי log.info() , log.debug()

- פונקציית Synchronized:
  - @Synchronized: אנוטציה על פונקציה. מגדיר בלוק synchronized על מנעול שהlombok יוצר.

- Delegate Pattern:
  - @Delegate: אנוטציה על משתנה.
    במידה ונרצה לממש את ה Delegate pattern, לפי הpattern נרצה להשתמש באובייקט קיים בשביל התנהגות קיימת, ולהרחיב את ההתנהגות שלו על ידי פונקציונליות נוספת. לכן נגדיר את האובייקט שמרחיבים עם אנוטציה @Delegate, וזה ייצור את הפונקציות המתאימות.