תכנות ותכן מונחה עצמים
046271

<u>דו"ח הגשה – תרגיל בית רטוב 0</u>

מתן צחור 208936989

פז וולף 206555138

שאלה 1

1. **Introduction:**
This document serves as our definition of “Good Coding Style” for code written in Java. A Java source file is described as written in “Good Coding Style” if and only if it adheres to the rules here.

1.1. Terminology Notes:
1.1.1. The term class refers to any “ordinary” class, enum, interface.
1.1.2. The term member refers to any nested class, variable, method or constructor within a class.

2. **Source File Structure:**
2.1. File name - Java source file includes only a single top-class, and the name of the file matches this class and ends with .java.
2.2. Imports – Packages and Imports will be shown first in the file, and will be separated by a single empty line. The class declaration will be separated from them by 2 empty lines.
2.3. Class order of content:
2.3.1. Logical order – All members will be ordered in a logical order (and not necessarily from oldest to newest). And will be ordered in the following order top to bottom: Classes, enums and constants, variables, static methods, methods.
2.3.2. Overloads – Methods with same name will be group together, and methods with relation and similar behavior will be grouped together.

3. **Naming conventions:**
Identifiers in our code will use only ASCII letters, digits and underscore. Identifiers will be meaningful and descriptive. Avoid using shorten versions of the actual word.

3.1. Class names are written in UpperCamelCase, and will usually be nouns or phrases, for example TextAnalyzer.
3.2. Constants will be written in UPPER_SNAKE_CASE, all in capital letters and each word separated by underscore.
3.3. Fields, Variables and Parameters will be written in lowerCamelCase, and will be with meaningful naming.
3.4. Methods are written in lowerCamelCase, and will usually start with a verb, for example printList.

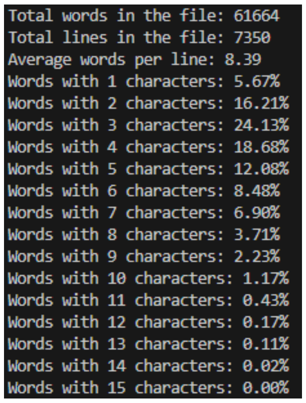
4. **File Formatting:**
4.1. Code will be written in a clear, consistent and logical manner.
4.2. Lines of code will be limited to 120 characters per line.
4.3. Each statement is written in a different line.
4.4. Each time a new block or block-like construct is opened, it is indented by a ‘tab’. When the block ends, the indent returns to the previous level. This applies to comments aswell.
4.5. Brackets are always used for blocks, even for empty ones.
No line break before opening brace, and always line break after opening brace.
Line break before and after closing brace.
Line break after every logical block. For example, in a if-else blocks sequence, the will be no line break between the if and else blocks, but the will be a line break after the else block.
In empty blocks, both braces can be in the same line as the statement, for example: void doNothing() {}.
4.6. Line break between consecutive members or initializers of a class: fields, constructors, methods, nested classes, static initializers, and instance initializers.
4.7. A single blank line may also appear anywhere it improves readability.
4.8. White space
4.8.1. Operators are to be surrounded by ‘space’s. For example:
salePrice = price * saleFactor.
4.8.2. Listed Variables or Parameters separated by ‘,’ followed by ‘space’.

5. Documentation and comments:
5.1. Use Javadoc-style comments to document classes, methods, and fields before the block. Documentation will include the following statements (as necessary):
@requires: mention any prerequisites that the input should meet.
@modifies: list any class members or passed arguments that might be modified.
@effects: provide a brief of the methods’ functionality, including expected output and modifications.
@return: return value (might be unnecessary if @effects is used).
@param: explains the role of each expected parameter (might be unnecessary if @effects is used).
@throws: list exceptions that might be thrown.

שאלה 2

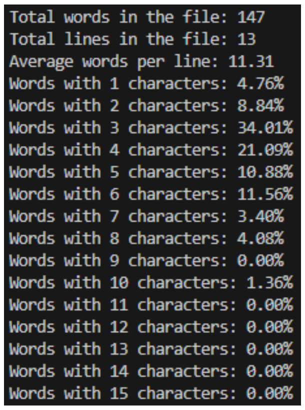
מצורפים הפלטים מריצת התכנית עם קבצי הקלט הנתונים:

1. עבור הקובץ Grapes.txt:



```
Total words in the file: 61664
Total lines in the file: 7350
Average words per line: 8.39
Words with 1 characters: 5.67%
Words with 2 characters: 16.21%
Words with 3 characters: 24.13%
Words with 4 characters: 18.68%
Words with 5 characters: 12.08%
Words with 6 characters: 8.48%
Words with 7 characters: 6.90%
Words with 8 characters: 3.71%
Words with 9 characters: 2.23%
Words with 10 characters: 1.17%
Words with 11 characters: 0.43%
Words with 12 characters: 0.17%
Words with 13 characters: 0.11%
Words with 14 characters: 0.02%
Words with 15 characters: 0.00%
```

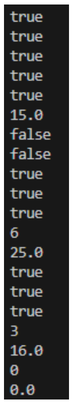
2. עבור הקובץ Engine.txt:



```
Total words in the file: 147
Total lines in the file: 13
Average words per line: 11.31
Words with 1 characters: 4.76%
Words with 2 characters: 8.84%
Words with 3 characters: 34.01%
Words with 4 characters: 21.09%
Words with 5 characters: 10.88%
Words with 6 characters: 11.56%
Words with 7 characters: 3.40%
Words with 8 characters: 4.08%
Words with 9 characters: 0.00%
Words with 10 characters: 1.36%
Words with 11 characters: 0.00%
Words with 12 characters: 0.00%
Words with 13 characters: 0.00%
Words with 14 characters: 0.00%
Words with 15 characters: 0.00%
```

שאלה 3

א. להלן פלט מתוכנית הבדיקה הפשוטה שלנו:



```
true
true
true
true
true
15.0
false
false
true
true
true
6
25.0
true
true
true
3
16.0
0
0.0
```

ב. המימוש הראשון שבו בחרנו הוא סכימת ערכי ה-$Volume$ בעת הפעלת המתודה $getVolume()$. היתרון של מימוש זה הוא שהמתודה לא מושפעת מ-$mutators$ של המחלקה $Ball$, ולא תחזיר תשובה שגויה אם המשתמש משנה כדורים שכבר הוכנסו למבנה, אך בעל סיבוכיות זמן ריצה גבוהה יותר.
לעומת זאת, המימוש השני בעל סיבוכיות זמן ריצה נמוכה יותר עבור המתודה $getVolume()$, אך הפעלת $mutators$ חיצוניים ישברנו את נכונות המתודה.

ג.

1. אין צורך לשנות את מפרט המתודה $add()$, מכיוון שהמפרט הנתון לא מתייחס למקרה זה והוא עדיין תקף.
2. במימוש שלנו, אנו מטפלים במקרה שבו הוכנס אובייקט שהוא $null$, ולכן אין צורך בשינויים לאחר הוספת פסקת ה-$@requires$ הנתונה.
3. המפרט החדש חלש יותר מהמפרט הקודם מכיוון שהוא דורש דרישות "קשוחות" יותר על הקלט.

שאלה 5

א. מפרט עבור שתי המתודות:

@requires – arr != null
@modifies - nothing
@effects – if val is in arr returns an index in the array of it’s appearance, otherwise returns a value outside of the array indices range.

ב. מפרט חזק יותר שיתאים ל-findLast:

@requires – arr != null
@modifies - nothing
@effects – if val is in arr returns it’s last index of appearance, otherwise returns -1.