

חוברת סטנדרטים

בסמ"ח מסלול אלפא

שקי חול



מבוא

כידוע, בתהליך פיתוח יש להגדיר סטנדרטים במטרה להקנות הרגלי עבודה נכונים שיעזרו להבנת הקוד ולשמירה על אחידות, בכדי לאפשר עבודה נוחה ומהירה יותר. חוברת זו מגדירה את הסטנדרטים על פיהם נפתח ונכתוב קוד בשפת java. שימו לב כי סטנדרטים אלה מסתמכים על סטנדרטי הפיתוח ש-google מגדירים.

ייתכן ובעת קריאת חוברת זו תתקלו בנושאים שטרם למדתם, ולכן יש להשתמש בחוברת זו במקביל לתהליך הלמידה שלכם בנושאים השונים, ובכל פעם להתרכז בסטנדרטים הרלוונטיים. זכרו! הקוד שלכם מייצג אתכם ומעיד על רמתכם המקצועית. קוד סטנדרטי הוא קוד קריא וברור.

הסטנדרטים של google:

אנו נשתמש בסטנדרטים המוגדרים של google – Google Java Style Guide. ניתן להגיע לסטנדרטים בחיפוש בגוגל או דרך הקישור הבא:

<https://google.github.io/styleguide/javaguide.html>

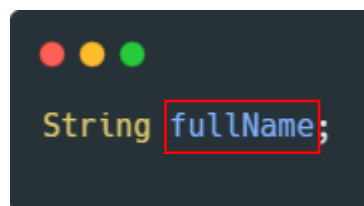
בחוברת הזו יהיו דגשים שקיימים גם באתר.

שמות

נקפיד על שימוש בשמות משמעותיים וקריאים – שמות שיכתבו באופן שיסביר את הקוד ויקל על הבנתו. השמות בהם נשתמש צריכים להסביר את הקוד למי שעתיד לקרוא אותו.

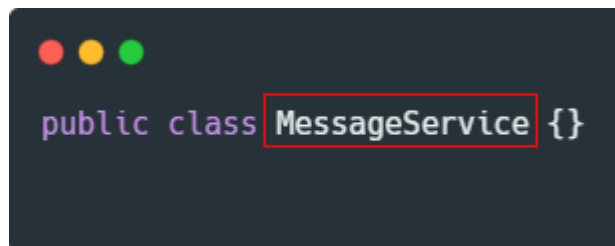
קיימות כמה צורות כתיבה עיקריות לכתיבת שמות ב-java:

- lowerCamelCase – נכתב באותיות קטנות, כאשר האות הראשונה בכל מילה מלבד הראשונה תהיה אות גדולה. אות גדולה מפרידה בין מילים שונות. לדוגמא:



```
String fullName;
```

- UpperCamelCase - נכתב באותיות קטנות, כאשר האות הראשונה בכל מילה כולל הראשונה תהיה אות גדולה. אות גדולה מפרידה בין מילים שונות. לדוגמא:



```
public class MessageService {}
```

- UPPER_CASE – נכתב באותיות גדולות עם קו תחתון המפריד בין מילים שונות. לדוגמא:



```
int EXIT_CODE;
```

יש כללים שונים לשמות של אלמנטים שונים. נעקוב אחרי כללים אלו כדי לשמור על קוד סטנדרטי וקריא.

מחלקות:

- שמות מחלקות יכתבו ב-UppercamelCase.
- שם המחלקה יהיה שם עצם.

```
class TextMessage {  
    ...  
}
```

משתנים ופרמטרים:

- אין לרשום את סוג הטיפוס (int, double, char ועוד...) בשם המשתנה - ניתן להבין את סוג הטיפוס מההכרזה על המשתנה ואין צורך בו בשם המשתנה עצמו. (למשל: לא נקרא למשתנה intSum)
- שמות משתנים יכתבו ב-lowerCamelCase.

```
int numOfTrophies;
```

קבועים:

- קבוע הוא אלמנט אשר לא משתנה לאורך הקוד - מוגדר כ-final.
- שמות קבועים יכתבו ב-UPPER_CASE.

```
final int MIN_HEIGHT;
```

תקנים חזותיים

עבודה עם סוגריים מסולסלים {} (Egyptian Brackets) ב-java:

- פתיחת הבלוק תבוא ללא ירידת שורה – באותה השורה, עם תו רווח אחד אחרי המילה השמורה, ואחריה תבוא ירידת שורה.
- סגירת הבלוק תבוא בשורה חדשה נפרדת.

דוגמא:

```
for (int count = 0; count < sum; count++) {  
    ...  
}
```

- אחרי סגירת בלוק תבוא שורת רווח לפני התחלה של קטע קוד חדש.

דוגמא:

```
for (int count = 0; count < sum; count++) {  
    ...  
}  
  
if (a < 3) {  
    ...  
}
```

- בין סגירת בלוק לסגירת בלוק לא תבוא שורת רווח מפרידה.

דוגמא:

```
for (int count = 0; count < sum; count++) {  
    if (a < 3) {  
        ...  
    }  
}
```

- אם לאחר סגירת הבלוק באה מילה שמורה שממשיכה את הבלוק הקודם, כמו if ו-else, מילה שמורה זו תבוא לאחר סגירת הבלוק ללא ירידת שורה (היא תהיה באותה השורה עם הסוגריים המסולסלים הסוגרים – וביניהם תו רווח).

דוגמא:

```

if (a < 3) {
    ...
} else {
    ...
}

```

- בכל פסוקית (for, if, else) נשתמש בסוגריים מסולסלים – גם אם מדובר בסוגריים המכילים פקודה אחת בלבד.

עימוד ותיחום:

- כל בלוק שיפתח יהיה מועמד טאב פנימה ביחס לבלוק בו הוא נמצא.
דוגמא:

```

public static void main(String[] args) {
    int taleLength = 10;
    if (taleLength > 5) {
        System.out.println("Mewo");
    } else {
        System.out.println("Hav");
    }
}

```

- כל פקודה תהיה בשורה נפרדת.
- אורך שורה תהיה עד 100 תווים (למעט שורות import/package בהן לא נבדוק את העניין) כללים לירידת שורה:
 - פיצול שורה יגיע לפני אופרטור (תהיה ירידת שורה אחריו).

```

int aVarienWithALongName = (aVarienWithEvenALongerName1 + aVarienWithEvenALongerName2)
    + aVarienWithEvenALongerName3;

```

- במידה ויש לולאות for של יותר מ-100 תווים, נפצל כל חלק בלולאה באותו הקו בדיוק.
לדוגמא:

```

for (int cityLocation = 0;
    cityLocation < AMOUNT_OF_CITIES;
    cityLocation++) {
}

```

ריווח:

- תהיה הפרדה של תו רווח בין מילים שמורות (if, for) וכדומה) לסוגריים שבאים אחריהן. לדוגמא:

```

if (numOfTrophies > maxNumOfTrophies) {
    ...
}

```

- תהיה הפרדה של תו רווח בין מילים שמורות (if, else) וכדומה) לסוגריים המסולסלים שקודמים להם.
- תהיה הפרדה של תו רווח בין סגירת סוגריים או מילה שמורה (if, else, for) וכדומה) לבין הסוגריים המסולסלים שבאים אחריהן. לדוגמא:

```

if (a < 3) {
    ...
} else {
    ...
}

```

- תהיה הפרדה של תו רווח משני הצדדים של אופרטור לוגי (>, <, &&, == וכדומה).
לדוגמא:

```
if (MTANumOfTrophies > MHFCNumOfTrophies) {  
    return "Maccabi is the bigger team :);  
}
```

- אחרי פסיק או נקודה פסיק יבוא תו רווח.
לדוגמא:

```
for (int count = 0; count < sum; count++) {  
    ...  
}
```

שורת רווח:

- תהיה שורת רווח לפני return.
לדוגמא:

```
if (playerNumOfTrophies > minNumOfTrophies) {  
    int playerSalary = (playerNumOfTrophies * salary) - tax;  
      
    return playerSalary;  
}
```


- תהיה שורת רווח לפני פונקציה (במידה ויש הערה, שורת הרווח תגיע לפנייה).

לדוגמא:

```
public static int calculateSum(int a, int b) {  
    return a + b;  
}  
  
public static void sayHelloTo(String name) {  
    System.out.println("Hello " + name);  
}
```

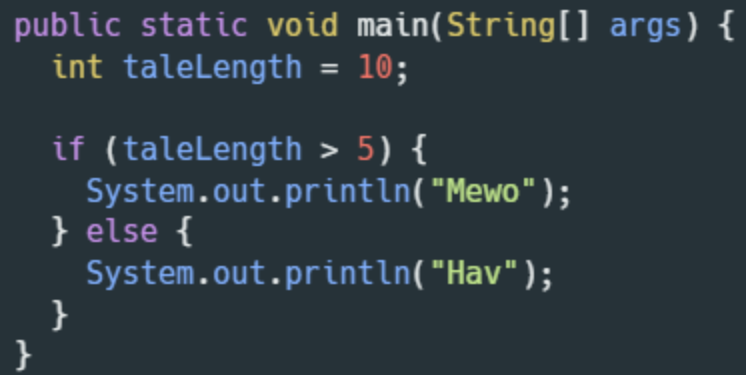
- בין סגירת בלוק לסגירת בלוק לא תגיע שורת רווח.
- בין סגירת בלוק לקטע קוד תגיע שורת רווח.

לדוגמא:

```
if (a > b) {  
    for (int count = 0; count < a; count++) {  
        b++;  
    }  
}  
  
int sum = a + b;  
System.out.println(sum);
```

- אחרי הצהרת מחלקה לא תבוא שורת רווח.

לדוגמא:



```
public static void main(String[] args) {  
    int taleLength = 10;  
  
    if (taleLength > 5) {  
        System.out.println("Mewo");  
    } else {  
        System.out.println("Hav");  
    }  
}
```

תיעוד

אנחנו כותבים את הקוד שלנו בגישה של clean code. כלומר, אנחנו נשאף שהקוד שלנו יהיה קריא ויסביר את עצמו, כך שאנחנו נוכל להשתמש במינימום הערות.

הערות טובות:

- Informative comments, הערות המרחיבות את המידע על קטע קוד מסוים (לדוגמא - אזכור של אלגוריתם קיים).
- Clarification, הבהרה לגבי משהו שהוא לא בהכרח קטע טבעי בקוד (לדוגמא - שימוש בקבוע מסוים)
- Warning of consequences, תיעוד על קוד שיכול ליצור בעיה בהמשך פיתוח בעתיד.

קוים מנחים:

- יש להסביר חלקים לוגיים בתוכניות או פקודות מסובכות ולא שגרתיות.
- נכתוב הערה אחת למספר שורות המבצעות מטלה אחת משותפת.
- הערות יכתבו באנגלית.
- לאחר פתיחת הערה יגיע תו רווח וההערה תתחיל באות גדולה.

דוגמה:

```
public static double calculateAvgGrade(int numOfStudents) {
    Scanner scanner = new Scanner(System.in);

    // Avg calculation
    int sumGrades = 0;

    for(int currentStudents = 0; currentStudents < numOfStudents; currentStudents++) {
        System.out.println("Enter student grade:")
        sumGrades += scanner.nextInt();
    }

    double avgGrade = sumGrades / numOfStudents;

    return avgGrade;
}
```

דגש חשוב!

קטע קוד זה לא לגמרי משקף מצב שבו נרצה להשתמש בהערה (הוא פשוט ומסביר את עצמו טוב). הוא מהווה דוגמה פשוטה שמסבירה את הכתיבה והרעיון הכללי של מתי נשתמש בהערות.

דגשים נוספים

משתנים:

- כל משתנה יוגדר בנפרד, בשורה משלו.

לדוגמא:

```
int numA;  
int numB;
```

- נשאף להגדיר משתנים כמה שיותר קרוב לשימוש הראשון בהם (לא בתחילת הבלוק – אלא אם יש בזה צורך).

- נשאף לאתחל את המשתנה מוקדם ככל האפשר – נעדיף לאתחל את המשתנה בהגדרתו.
לדוגמה:

```
public static void main(String[] args) {  
    int num1 = 10;  
    int num2 = 20;  
  
    int sum = num1 + num2;  
    System.out.println("The sum of " + num1 + " and " + num2 + " is " + sum);  
  
    String str = "Hello World!";  
    if (str.contains("World")) {  
        System.out.println("The string \"" + str + "\" contains the word \"World\"");  
    } else {  
        System.out.println("The string \"" + str + "\" does not contain the word \"World\"");  
    }  
}
```

לולאות:

- שם מונה הלולאה יעמוד בתנאי הקריאות (לא נשתמש בשמות כמו i/j/z וכו').

לדוגמה:

```
for (int count = 0; count < sum; count++) {  
    ...  
}
```

- לא נשנה את מונה הבקרה (מונה הלולאה) בתוך הקטע קוד שבתוך הלולאה.

- לא נשתמש במונה הלולאה מחוצה לה.
- לא נשתמש באותו משתנה ל-2 תפקידים שונים.

Google-Java-Format

עיקר העיסוק שלנו צריך להיות כתיבת הקוד עצמו ולא הנראות שלו מבחינת עימודים ריווח וכדומה. על כן, אנחנו יכולים להשתמש ב-formatter שתפקידו לעזור לנו לעמך את הקוד שלנו על פי סטנדרטי הפיתוח של google.

כיצד נוסיף את הפורמטר?

1. ניכנס ל-Settings (ניתן לעשות זאת בעזרת לחיצה על `ctrl+alt+s`).
 2. ניכנס ל-Plugins.
 3. נחפש google-java-format.
 4. נוריד את plugin (נלחץ על install).
- עד כה הוספנו את ה-Plugin, אך הוא לא בשימוש (מוגדר כ-disabled by default)
- כדי להפעילו נבצע את הפעולות הבאות:
5. נחפש ב-settings את google-java-format settings.
 6. נסמן ב-V את ה-checkbox של Enable google-java-format.
- כדי להפעיל את פעולותיו בכל פרויקט חדש שנפתח נבצע את הפעולות הבאות:
7. נלחץ על file נכנס ל-other setting -> settings for new project.
 8. נחפש את google-java-format settings.
 9. נסמן ב-V את ה-checkbox של Enable google-java-format.

כעת, כאשר נבצע את קיצור המקשים `Ctrl+Alt+L` – יתבצע פירמוט של הקוד כך שיתאים לסטנדרטי העימוד והריווח של google.

לדוגמא:

קטע הקוד הבא

```
public class Main {
    public static void main(String[] args) {
        int firstNumber = 1;
        int secondNumber = 1;

        if(firstNumber > secondNumber){
            System.out.println("The first number is bigger");
        }else if(secondNumber > firstNumber) {
            System.out.println("The first number is bigger"); }else{ System.out.println("The numbers are equal!");}
    }
}
```

יהפוך לקטע הקוד הבא

```
public class Main {
    public static void main(String[] args) {
        int firstNumber = 1;
        int secondNumber = 1;

        if (firstNumber > secondNumber) {
            System.out.println("The first number is bigger");
        } else if (secondNumber > firstNumber) {
            System.out.println("The first number is bigger");
        } else {
            System.out.println("The numbers are equal!");
        }
    }
}
```