

עשר השגיאות הנפוצות של מתכנתים מתחילים

אם אתם חדשים בתחום התכנות, ועושים את צעדכם הראשונים בשפת תכנות – ברוכים הבאים. שפת תכנות היא השפה שבאמצעותה אנו אומרים למחשב מה לעשות ומעבירים לו את הוראותינו. שפת תכנות דומה בכמה מובנים לשפה אנושית שבה אנו מדברים וכותבים – כמו לשפה אנושית, גם לשפת תכנות יש תחביר (syntax) שקובע מה המבנים החוקיים בשפה, אוצר מילים, וסמנטיקה, שנותנת משמעות לצירופי המילים לפי חוקי התחביר. אבל בשפה אנושית, הנמען הוא בן אדם אחר, שקורא את מה שכתבנו או שומע את מה שאנחנו מדברים. לעומת זאת, בשפת תכנות הנמען הוא המחשב, או יותר נכון **הקומפיילר (Compiler)** שאמור "לקרוא" את מה שכתבנו ולבצע. אמנם בשפה אנושית ישנם כללי דקדוק ותחביר, אולם אנחנו מאוד סובלניים לשגיאות דקדוק, ובדור"כ אנחנו מבינים אחד את השני גם כאשר אנחנו לא מדברים תמיד לפי חוקי השפה. בניגוד לשפה אנושית, שפת תכנות היא לחלוטין לא סובלנית לשגיאות. הקומפיילר דורש שכל מה שנכתב יעמוד במדויק בחוקי התחביר של השפה, וכל שגיאה, ולו הקטנה ביותר, לא תאפשר הרצת התוכנית.

מתכנתים מתחילים בדור"כ נוהגים לעשות שגיאות שחוזרות על עצמן, שגיאות שנובעות בעיקר מהקושי להתרגל למבנה הנוקשה של השפה ולחוקים שלה, ומהקושי להתרגל לעובדה ששפת התכנות לא סלחנית לאף שגיאה, גם לא הקטנה. הקומפיילר לא יכול "להבין את הכוונה" שלנו אם מה שכתבנו לא עומד בחוקים. במאמר זה אציג את השגיאות הנפוצות של מתכנתים מתחילים בשפת ג'אווה. חלקן, אגב, קורות עדיין גם למתכנתים מנוסים.

1. חוסר בסוגריים מסולסלים

הודעת הקומפיילר: '}' expected

תיאור הבעיה: בשפת ג'אווה, סוגריים מסולסלים משמשים לתחימת בלוקים בקוד – משפטי תנאי, לולאות, שיטות, מחלקות ועוד נתחמים כולם ע"י סוגריים מסולסלים. לכל סוגר פותח צריך להיות בן זוג סוגר, שחסרוננו יגרום לשגיאת קומפילציה. הסיבה לשגיאה הזאת היא שבדור"כ הקוד שנכתב בתוך הסוגריים ארוך, ולכן בזמן שעובר מרגע פתיחת הסוגריים המתכנת שוכח שהוא גם צריך לסגור אותם. הבעיה נהיית יותר חמורה כאשר התוכניות נהיות יותר גדולות, כיוון שאז כמות הסוגריים עולה בצורה משמעותית, והם גם נכנסים אחד בתוך השני, ככה שלמצוא את הסוגר החסר זאת בדור"כ משימה מייאשת וגוזלת זמן.

למשל, התבוננו בדוגמת הקוד הבאה. האם תוכלו לאתר איפה חסר הסוגר המסולסל?

```
public class Car {
    private int x1 = 0, y1 = 0;
    public void drive(int x) {
        if(x < 0) {
            if(x1 < 10) {
                x1++;
                y1 = x1 * x;
            }
            else {
                x1--;
                y1 = x1 * y1;
            }
        }
        else {
            x1 = y1 - x;
        }
    }
}
```

}
אגב, אותו כלל תקף גם לכל שאר הסימנים ה"זוגיים" בשפה – סוגריים (), סוגריים מרובעים [], סוגריים משולשים <>, גרשיים '' וגרש '.

הפתרון: ניתן למנוע את השגיאה הזאת במלואה ע"י הרגל פשוט וקל ליישום – בכל פעם שפותרים סוגריים מסולסלים, יורדים שורה, סוגרים אותה ורק אז כותבים את הקוד. כך מובטח שלא נשכח את הסוגריים והבעיה נפתרת במלואה. נכון שזה נשמע פתרון מטופש, אבל זה מסוג הדברים שלומדים "דרך הידיים" – ברגע שתבלו שעה מזמנכם באיתור סוגר נעלם, תבינו שחייבים להמנע מהבעיה הזאת מראש.

2. חוסר בנקודה פסיק בסוף פקודה

הודעת הקומפיילר: expected ';' ;

תיאור הבעיה: בשפת ג'אווה, הסימן נקודה-פסיק ';' מסמן "סוף פקודה". הקומפיילר מחפש את הסימן הזה כדי לדעת שפקודה הסתיימה. אם הקומפיילר מוצא פקודה שלא מסתיימת, תתרחש שגיאת קומפילציה. הסיבה לשגיאה היא פשוט הרגל – קשה לנו להתרגל שצריך לשים סימן בסוף פקודה, ולכן שוכחים אותו. שימו לב שבניגוד לפקודות שמבצעות משהו, הסימן נקודה-פסיק לא מבצע שום דבר, ולכן קל לנו לשכוח אותו.

הפתרון: פשוט להתרגל. כמו שאנחנו רגילים לשים נקודה בסוף משפט, ככה צריך להתרגל לסיים פקודה בנקודה-פסיק. ככל שמתרגלים יותר כתיבת קוד, ככה השגיאה הזאת נהיית נדירה יותר ויותר.

3. עודף בנקודה-פסיק

הודעת הקומפיילר: כל מיני, תלוי בהקשר. לפעמים זאת לא שגיאת קומפילציה, לפעמים השגיאה היא `Missing method body or declare abstract` כאשר הסימן נקודה-פסיק מופיע בסוף חתימת השיטה, לפעמים השגיאה היא `else without if` כאשר הסימן נקודה-פסיק מופיע אחרי תנאי שיש לו גם `else` ויכולות להיות גם הודעות אחרות.

תיאור הבעיה: כאמור בסעיף הקודם, הסימן נקודה-פסיק מסמן סוף פקודה. אולם ישנן פקודות שטווח ההשפעה שלהן הוא גדול. למשל, פקודת התנאי `if` מאפשרת לשאול שאלה ולבצע שורות קוד מסויימות כתלות בתשובה. במובן הזה, הפקודה `if` לא מתקיימת כפקודה בפני עצמה, אלא היא וכל הקוד שמתחתיה נחשבים ליחידה לוגית אחת. אותו הדבר קורה גם בהצהרה על שיטה – החתימה של השיטה היא רק ההתחלה, מתחתיה מגיע מימוש השיטה, שנחשב ליחידה לוגית אחת יחד עם החתימה. פקודות מהסוג הזה לא אמורות להסתיים בנקודה-פסיק, כי הן בעצם עוד לא הסתיימו. הבעיה הזאת נגרמת בדרך"כ אחרי שהמתכנת התרגל לרשום נקודה-פסיק, הוא רושם את הסימן בכל מקום, גם כשלא צריך.

שימו לב שלפעמים תוספת הסימן לא תגרום לשגיאת קומפילציה, אלא לשגיאה לוגית, כלומר הקוד ירוץ, אבל לא יבצע את מה שהתכוונתם שהוא יבצע. למשל, התבוננו בקטע הקוד הבא, מה ערכו של `y` בסוף הקטע?

```
int x = 2, y = 0;  
if(x == 8);  
    y = 5;
```

שימו לב לנקודה-פסיק בסוף התנאי. במקרה הזה זאת לא שגיאת קומפילציה, אבל הקומפיילר חושב שפקודת ה-`if` מסתיימת ברגע שהוא רואה את הנקודה-פסיק, ולכן המשפט `y=5` בעצם לא קשור לתנאי. לא משנה אם התנאי מתקיים או לא מתקיים, ההצבה שבאה אחריו תקרה בכל מקרה. לזה קוראים שגיאה לוגית – אנחנו חושבים שמשוואה אחת יקרה, אבל בפועל כתוצאה מחוסר תשומת לב, משהו אחר קורה וגורם לתוכנית להתנהג באופן לא צפוי.

הפתרון: שוב, כמו בסעיף הקודם – צריך להתרגל. ככלל אצבע אפשר להגיד שכל פקודה שאחריה נפתחים סוגריים מסולסלים לא אמורה להסתיים בנקודה-פסיק. אפשר גם להפעיל הגיון בריא – אם הפקודה משפיעה על שורות שבאות אחריה, כנראה שהיא לא צריכה להסתיים בנקודה-פסיק.

4. שימוש בסימן = במקום == במשפט תנאי

הודעת הקומפיילר: incompatible types

תיאור הבעיה: במשפטי תנאי, פעולת ההשוואה צריכה להיעשות ע"י הסימן == שמייצג את השאלה "האם שווה?". לעומתו, הסימן = הוא סימן הצבה שמורה למחשב להציב את הערך שבצד ימין למשתנה שבצד שמאל, ומובן שאין פה שאלה. לכן כתיבה של תנאי בצורה הבאה:

```
if(x = 1)
```

בעצם מנסה לבצע משפט הצבה בתוך משפט תנאי. כיוון שמשפט התנאי חייב לקבל שאלה בוליאנית בתוך הסוגריים, יש פה חוסר התאמה בין הטיפוסים – הקומפיילר מצפה שבתוך הסוגריים יהיה איזשהו ערך בוליאני, אבל בפועל מקבל ערך אחר. הסיבה לשגיאה היא הקושי להפנים את העובדה שיש הבדל בין השמה לבין שאלה. שימו לב שבמקרה אחד הכתיבה הזאת לא תיתן שגיאת קומפילציה, למשל –

```
boolean b = false;
```

```
if(b = true)
```

```
System.out.println("OK");
```

הקומפיילר מצפה לקבל ערך בוליאני בתוך התנאי, אבל זה בדיוק מה שהוא מקבל! לכן בתוך התנאי מבצעים השמה של הערך true למשתנה b, התנאי מתקיים ובכך קיבלנו שגיאה לוגית חמורה – התנאי התקיים למרות שהוא לא אמור היה להתקיים, ובנוסף גם שינינו את ערכו של המשתנה b.

הפתרון: כרגיל, להתרגל להבדל בין = לבין ==. השגיאה מאוד ברורה ומיידית, ככה שקל לאתר אותה ולתקן אותה. יש אנשים שהתרגלו לכתוב את התנאי בצורה הזאת:

```
if(true == b)
```

זה פותר את הבעיה של משתנה בוליאני, כיוון שאפשר לשאול "האם b שווה ל-true" ואפשר גם "האם true שווה ל-b". מצד שני, אם היינו כותבים בטעות

```
if(true = b)
```

היתה פה שגיאת קומפילציה, כיוון שהצבה מחייבת לרשום את שם המשתנה שאליו מציבים מצד שמאל של סימן השוויון, וכך היינו רואים שיש פה בעיה.

5. בלבול בין אותיות גדולות לקטנות

הודעת הקומפיילר: cannot find symbol, package does not exist, למשל, class expected ועוד.

תיאור הבעיה: שפת ג'אווה היא שפה שמבדילה בין גודל האות (case sensitive language) ומבחינת השפה האות A והאות a הן שני דברים שונים. לכן אם נגדיר משתנה בשם a אבל בקוד ננסה להתייחס אליו כאל A הקומפיילר לא יכיר את המשתנה המדובר. אותה בעיה קיימת גם בפקודות של השפה – כל המילים השמורות בשפה כתובות באותיות קטנות, ולכן אם ננסה לרשום int במקום int הקומפיילר לא יכיר את הפקודה.

הפתרון: להיות מדויקים ולשים לב למה שכותבים. כדאי גם לזכור שכל הפקודות בשפה הן באותיות קטנות, וכדאי גם לרשום לפי המוסכמות של השפה, שמתארות את כללי מתן השמות למשתנים, מחלקות, שיטות ועוד, ככה מתרגלים לסגנון הכתיבה ועושים פחות טעויות. בנוסף, אם כותבים בסביבת עבודה מתקדמת כדוגמת Eclipse או NetBeans, ניתן לראות שסביבת העבודה צובעת בצבעים שונים את המילים השמורות בשפה, ולכן אפשר לראות בצורה ויזואלית מתי טעינו בכתיבה.

6. שימוש במשתנה שלא הוגדר

הודעת הקומפיילר: cannot find symbol variable

תיאור הבעיה: הקומפיילר דורש שנגדיר כל משתנה לפני שמשתמשים בו. לכן אם הקומפיילר רואה שימוש בשם משתנה שלא מוכר (לא הוגדר קודם), הוא לא יודע מי המשתנה הזה ומה הטיפוס שלו, ולכן הוא מוציא הודעת שגיאה.

הפתרון: לזכור – כל דבר שמשתמשים בו צריך להיות מוגדר קודם.

7. שימוש במשתנה שלא אותחל

הודעת הקומפיילר: variable might not have been initialized

תיאור הבעיה: כאשר מגדירים משתנה, מקבלים מהמחשב תא בזיכרון. תא זה מכיל ערכים קודמים שהיו בו (תאי הזיכרון משמשים תוכניות רבות במהלך עבודת המחשב). כדי להיות בטוחים שתא הזיכרון שקיבלנו "נקי" ומכיל ערכים שאנחנו אחראים להם, יש לאתחל את המשתנה, כלומר להציב לתוכו ערך. הקומפיילר מוודא שלפני כל שימוש במשתנה, יש בתוכו ערך מאותחל, כלומר ערך שהמתכנת קבע, ולא ערך "זבל" – כלומר ערך שנשאר בזיכרון משימושים קודמים. למשל, התבוננו בקוד הבא:

```
int x, y;  
x = y + 2;
```

בקוד מוגדרים שני משתנים – x, y . אנו מנסים להציב לתוך x את הערך של y ועוד 2. אולם ל- y אין כרגע ערך, ולכן הקומפיילר יתן את ההודעה. שימו לב, לעומת זאת, שהקומפיילר לא יתן את ההודעה על ההצבה ל- x כיוון שבעצם ביצוע (או נסיון ביצוע) ההצבה למשתנה מהווה אתחול למשתנה. כלומר, השגיאה נובעת מנסיון להשתמש בערך של משתנה, לפני שלמשתנה יש ערך כלשהו.

הפתרון: פרקטיקה שיכולה לעזור היא להתרגל להגדיר משתנה ולאחל אותו באותה שורה, למשל –

```
int x = 0, y = 0;
```

בצורה הזאת אנחנו מבטיחים שלמשתנים שהגדרנו תמיד יש ערך. אחרי שמתרגלים לכתיבה, אפשר לוותר על אתחולים לא רלבנטיים. כלומר, אם המשתנה x אמור לקבל הצבה בהמשך הקוד, אין טעם לאתחל אותו באפס בהגדרה (למרות שזה לא מזיק).

8. הצבה של ערכים לא תואמים מבחינת טיפוס נתונים

הודעת הקומפיילר: possible loss of precision

תיאור הבעיה: בג'אווה, כאשר מציבים ערך למשתנה, הטיפוסים של הערך המוצב חייב להיות זהה לטיפוס של המשתנה אליו מציבים אותו. אם הטיפוס אינו זהה, הקומפיילר יבצע פעולה שנקראת **המרה (casting)**, כלומר, יהפוך את הערך המוצב "בכוח" להיות כמו הטיפוס של המשתנה. אבל המרה כזאת עלולה להסתיים בתוצאות לא צפויות כאשר אין אפשרות פרקטית להפוך סוג אחד לסוג אחר. למשל:

```
int x;  
double y = 2.6;  
x = y;
```

המשתנה x מוגדר כ-int ולכן הוא תופס תא זיכרון שיועד להכיל אך ורק מספרים שלמים. המשתנה y לעומתו מוגדר כ-double והוא יכול להכיל מספרים ממשיים (כמו שהוא באמת מכיל בדוגמא). איך נציב מספר ממשי לתא זיכרון שלא יכול להכיל אותו? אי אפשר, והמחשב יאלץ לבצע פה מניפולציה של המספר הממשי כדי "שיתאים" לתא זיכרון מסוג int. ברור שכתוצאה מהמניפולציה, הערך המקורי של המספר לא ישמר, ולכן הקומפיילר מודיע את ההודעה. שימו לב שההיפך דווקא אפשרי, אין בעיה להכניס מספר מסוג int לתא מסוג double.

הפתרון: אין פתרון טוב, פשוט צריך לזכור שהמספרים בג'אווה מחולקים לסוגים שונים. השגיאה ברורה ומיידית, וקל לאתר אותה.

9. חוסר ברווחים, שורות חדשות וטאבים

הודעת הקומפיילר: אין, זאת לא שגיאת קומפילציה.

תיאור הבעיה: שפת ג'אווה אדישה לרוב הרווחים, שורות חדשות וטאבים. כלומר, אין חובה מבחינת השפה להשתמש בהם (למעט מקרים ברורים כמו למשל להפריד בין פקודות). אולם כתיבת קוד צפוף, בלי ריווח כלשהו, מקשה מאוד על קריאתו, ולכן גם מקשה על התחזוקה שלו ומהווה פתח טוב לשגיאות עתידיות. למשל, התבוננו בקוד הבא:

```
public class Tester{public static void main(String[] args){System.out.println("Hello");}}
```

הקוד עובר קומפילציה ורץ, אבל בוודאי שלא קל לקרוא אותו, ובוודאי שהבעיה תהיה חמורה הרבה יותר כאשר התוכנית תכיל הרבה יותר שורות קוד.

הפתרון: לאמץ כמה כללי סגנון חשובים שיעשו את החיים לנוחים יותר – כל פקודה חדשה תיכתב בשורה חדשה. בפרט, אחרי כל נקודה-פסיק יש לרדת שורה (חוץ מנקודה-פסיק שמופיע בהגדרת לולאת for). בכל פעם שפותחים סוגריים מסולסלים יש לרדת שורה ולהשתמש בטאב. כל הקוד בתוך הסוגריים המסולסלים בעצם יהיה טאב אחד ימינה מהסוגריים. גם אחרי סגירת הסוגריים יש לרדת שורה. בלוקים שונים של קוד שמתעסקים בדברים שונים יש להפריד בשורת רווח. כדאי להשתמש ברווחים לפני ואחרי אופרטורים חשבוניים ופעולות הצבה. ובכלל, נסו לכתוב קוד קריא ומרווח.

10. עודף רווחים, שורות חדשות וטאבים

הודעת הקומפיילר: אין, זאת לא שגיאת קומפילציה.

תיאור הבעיה: ההפך מהבעיה הקודמת – כיוון שהשפה אדישה לרווחים, היא אדישה גם לקיומם. מתכנתים מסויימים נוטים לנצל את העובדה הזאת ולהשתמש ברווחים ושורות חדשות באופן מוגזם שלמעשה פוגע בקריאות של התוכנית. למשל –

```
System  
.  
out  
.  
println  
(  
"hello"  
)  
;
```

הקוד הזה שמתפרש על פני 9 שורות הוא בעצם פקודת ההדפסה הרגילה של ג'אוה, אבל קשה מאוד לקרוא אותה בגלל שצריך לעקוב אחרי הרבה שורות.

הפתרון: כמו בסעיף הקודם, הקפידו על כללי עימוד ברורים. ככלל, אף פעם אל תכתבו את סימן הנקודה-פסיק בשורה נפרדת, אלא תמיד בצמוד לפקודה אותה הוא סוגר. כמו כן כדאי לא לפצל פעולות חשבוניות לכמה שורות (אלא אם כן הן ארוכות מאוד) וכדאי לא לשים יותר מרווח אחד בין מילה למילה.

סיכום

במאמר הזה הצגתי עשר מהשגיאות הנפוצות ביותר בקרב מתכנתים מתחילים. רוב השגיאות נעלמות לאחר תרגול וכתובה של קוד בשפה, אולם בשלבים הראשונים של הכתיבה השגיאות עלולות להיות מתסכלות ולא מובנות. אני מקווה שמאמר זה יספק מדריך שימושי להתגברות על שגיאות אלו.

תרגיל

הנה תרגיל – בקוד שלהלן ישנן לא מעט שגיאות קומפילציה. האם תוכלו לאתר את כולן ולגרום לקוד לעבור קומפילציה? אני ממליץ להעתיק את הקוד לסביבת העבודה ולראות מה ההערות שהקומפיילר נותן. אם תצליחו לגרום לקוד לעבור קומפילציה, הרצת התוכנית תוציא על המסך הודעת הצלחה.

```
Public class Tester {
    public static void main(string[] args);
    {
        int count, x;
        long len = 0;
        count = 2;
        if(Count = x)
        {
            X = y - 1;
            count—
            if(count == y)
            {
                if(x > 0)
                    y = 8;
                else;
                {
                    y = 9;
                    x--;
                }
            }
            else
                count = count * 7;
        }
        else
        {
            count = len + 1;
        }
        System.out.println("At last – All compilation Errors were fixed!");
    };
};
```

שאלות? תגובות? תיקונים? כתבו לי לכתובת shay.tavor@gmail.com