שפות תכנות – תרגיל מספר 3 חלק יבש

תאריך הגשה: *14.12.17*

מגישים:

סהר כהן – 206824088

יובל נהון – 206866832

1. (a) מערכת הטיפוסים של Kotlin:
- שפת Kotlin היא Typed programming language,
- שפת Kotlin היא Strongly-typed (טיפוסית חזקה / נוקשה),
- מערכת הטיפוסים בשפת Kotlin אינה מפלה (אין "second-class" types),
- שפת Kotlin תומכת ב static typing וגם ב – dynamic typing, אבל קטע קוד סטנדרטי בשפה ישתמש בעיקר ב – dynamic typing: אין הכרזה על טיפוסי המשתנים.
- שפת Kotlin היא בעלת explicit typing(טיפוסיות מפורשת), כלומר – שם הטיפוס חייב להיות כתוב במפורש (כמו C, Java), בניגוד לשפה בעלת implicit typing כמו ML.
- רמת המורכבות של מערכת הטיפוסים בשפת Kotlin היא מורכבת: כשפה מונחית עצמים, קיימת תמיכה ב C'tors מורכבים במיוחד, ליצירת טיפוסים מורכבים.

(b) הטיפוסים Any, None, Unit קיימים ב – Kotlin:
i. Unit - טיפוס שמקבל אך ורק ערך יחיד. מתקבל, למשל, על ידי יצירת class ריק.
ii. None – טיפוס המציין שלאובייקט אין אף ערך. מתקבל, למשל, דרך enum ריק (ללא ערכים).
iii. Any - טיפוס-על: כל הטיפוסים האחרים בשפה הם תתי-טיפוסים שלו. מתקבל, למשל, ביצירת פונקציות/מחלקות גנריות המקבלות ערך מטיפוס גנרי "T" - טיפוס זה יכול להיות כל ערך.

2. **טיפוסיות שמית**: בהשוואת שני ערכים מורכבים, הם ייחשבו שווים אם ורק אם כל השדות של האחד שווים בהתאמה לשדות של השני, והם מאותו הטיפוס.
לדוגמה: השוואה של structs בשפת C (השוואה ערך-ערך).
טיפוסיות מבנית: בהשוואת שני ערכים מורכבים, הם ייחשבו לשווים אם הטיפוסים שלהם זהים מבחינת המבנה, וערכי השדות שווים בהתאמה.
לדוגמה: השמה של מצביע לטיפוס כלשהו להיות מצביע לטיפוס שונה בעל אותו המבנה, למשל עבור מצביע ל – long שעובר להצביע ל – int.

3. ההבדל בין טיפוס לערך הוא שטיפוס יכול להכיל בתוכו הרבה ערכים שונים, והרבה סוגים שונים של ערכים - ולעומת זאת, ערך לא מכיל בתוכו כלום חוץ ממה שהוגדר לו. במילים אחרות, טיפוס הוא אוסף של ערכים. ערך הוא ביטוי שאי אפשר לחשב אותו בצורה פשוטה יותר, בעוד שטיפוס יכול להיות ערך: למשל, מבני נתונים מורכבים המכילים בתוכם טיפוסים אשר נחשבים להיות ערכים של מבנה הנתונים, משום שלפי הגדרת המבנה: אין דרך יותר פשוטה לחשב את הטיפוסים. במצב כזה הטיפוס יכול להיות גם ערך.

4. ההבדל בין type aliasing לבין type branding הוא שב type aliasing נותנים לאותו אובייקט שם נוסף וקיימת ביניהם שקילות מבנית (ראו: סעיף 2). כלומר, הם להיות אותו הטיפוס ואפשר לבצע עליהם את אותן הפעולות. דוגמה: Typedef עבור struct בשפת C לא יוצר טיפוס חדש, אלא מקנה לטיפוס קיים שם נוסף.
לעומת זאת, ב type branding יוצרים טיפוס חדש על סמך טיפוס אחד או יותר והשימוש ב - type branding גורר שימוש בשקילות שמית (שוב ראו: סעיף 2) בין הטיפוסים הנוצרים: שני טיפוסים עם מבנים זהים יחשבו שונים אם הם לא מאותו טיפוס. ההבדל בין שני המושגים חשוב למשל בתכנית המייצגת חנויות, כאשר ישנם מוצרים זהים השייכים לחברות שונות. כלומר, הם זהים מבחינה מבנית אך לא שייכים לאותו "טיפוס", שכן הם נמצאים בבעלות חברה שונה, ולכן הם שונים.

5. שגיאת טיפוס היא שגיאה שבה חורגים מערך הטיפוס שמשתמשים בו, למשל עבור חריגה מתמטית: כאשר ישנה חלוקה ב – 0, א\ו כאשר מפעילים שורש על מספר שלילי, תוצאות פעולות אלה חורגות מטווח הערכים החוקי של הטיפוסים int או double וגורמים לשגיאת טיפוס. דוגמה נוספת היא מערכים: לכל מערך יש תחום אינדקסים חסום המוגדר מרגע האתחול, ולכן פנייה למערך עבור אינדקס היוצא מתחום הערכים החוקי הזה תהיינה לא

חוקית: שגיאה זו היא גם שגיאת טיפוס. ייתכן כי הקוד יתקמפל אם בשלב הקומפילציה עדיין אין חריגה מטווח הערכים של הטיפוס, והשגיאה תתגלה רק בזמן ריצה.

דוגמא לשגיאת טיפוס אשר מתגלה בזמן קומפילציה:

```
double num = 10 / 0; //Does not compile!
```

דוגמא לשגיאת טיפוס אשר מתגלה בזמן ריצה:

```
int num1 = 3, num2 = 3;
int result = 10 / (num2-num1); //Run-time error!
int arr[] = new int[3];
arr[num1] = num2; //Run-time error!
```

נציין גם שהדוגמאות המובאות הן דוגמאות משפות בסגנון C, C++, Java: שפות בעלות טיפוסיות סטטית. מכאן ששגיאות טיפוס **לא** יכולות להימנע על ידי אכיפה של טיפוסיות סטטית, ולכן, במידה רבה זוהי אחריותו של המתכנת שלא לבצע שגיאות מסוג זה.