

תוכנה 1 - הערות לחלק 2

שימוש בתו כשלם

בפעולות על מספרים, ג'אוה מתייחס לתווים כאל שלמים אי שליליים, וניתן להוסיף אותם למספרים, לחסר אותם ממספרים, וכולי. ניתן גם לחבר ולחסר מספרים מתווים, מה שמייצר תווים אחרים:

```
char c = 'A';
char d = c + 1; // d is 'B'
int i = 3 - d; // i = 3 - 66 = -63
```

טיפוסים של נקודה צפה

ג'אוה מספקת שני טיפוסים עבור משתנים שמכילים מספרים ממשיים בייצוג של נקודה צפה:

float 32 סיביות: 1 לסימן, 8 לחזקה (של 2), ו-23 לשבר

double 64 סיביות: 1 לסימן, 11 לחזקה, ו-23 לשבר

שני הטיפוסים מיועדים לייצוג בפורמט סטנדרטי בשם IEEE-754. משתנים בייצוגים הללו יכולים לייצג לא רק קבוצה של מספרים ממשיים, אלא גם

- אינסוף ומינוס אינסוף, שמייצגים מספר שמעבר לטווח של הייצוג או גבול של סדרה שמתכנסת לאינסוף. למשל, $-1.0/0.0 = -\infty$.
- "אינו מספר", NaN או Not a Number, שמייצג ערך או תוצאת חישוב לא מוגדרת, כמו $0.0/0.0$.
- אפסים חיוביים ושליליים, שמייצגים את הערך 0 או גבול של סדרה שמתכנסת ל-0 (מהכיוון החיובי או השלילי).

שוויון בין ערכי נקודה צפה

כאן יש הפתעות, אבל מסתתר מאחוריהן הגיון. לגבי מספרים רגילים שאינם 0, ההתנהגות פשוטה: == מחזיר true אם שני הערכים שווים, אחרת false. (שווים בדיוק! בגלל שגיאות עיגול, $1e20+1e0-1e20==0$, למשל)

לגבי אפסים, == מחזיר true אם שני הערכים הם אפס, גם אם אחד הערכים הוא 0.0 והשני הוא -0.0. הערכים הללו משקפים גבול של סדרה שמתכנסת ל-0 מהכיוון החיובי או מהכיוון השלילי. הסימן של האפס "זוכר" מהיכן הסדרה התכנסה ל-0, אבל לגבי האופרטור == חשוב לאן הסדרה התכנסה, לא מהיכן היא הגיעה. שוויון בין אפסים (ובין ערכי נקודה צפה באופן כללי) לא מלמד ששני הערכים השווים מתנהגים באופן זהה. למשל, אפסים מסיימים שונים מודפסים באופן שונה; ערך הביטוי $1.0/0.0$ הוא אינסוף אבל ערך הביטוי $1.0/-0.0$ הוא מינוס אינסוף.

אינסופים מתנהגים באופן פשוט יותר: אינסוף חיובי שווה לאינסוף חיובי ושלילי לשלילי, אבל אינסוף חיובי לא שווה לאינסוף שלילי.

אינו מספר (not a number, NaN), שונה מכל ערך, כולל מכל NaN. גם זה מפתיע, אבל הגיוני. כאשר משתנה מייצג ערך שאין יודעים מה ערכו, אי אפשר לומר שהוא שווה לערך של משתנה אחר, גם אם ערכו של המשתנה האחר אינו ידוע.

לסיכום, האופרטור == בודק לגבי ערכי נקודה צפה שוויון כמותי ולא שוויון בין הייצוגים של הערכים במחשב. ברוב המקרים, ההתנהגות של == היא ההתנהגות הרצויה באלגוריתמים נומריים.

בדיקת שוויון על ידי בדיקת זהות של עצמים מקובעים (ומחרוזות)

בדיקת שוויון בעזרת השירות equals היא בדרך כלל פחות יעילה מאשר בדיקת זהות של התייחסויות בעזרת ==. למשל, בדיקת שוויון בין מחרוזות בודקת את שתי המחרוזות תו אחר תו.

מחרוזות בג'אווה הן עצמים מקובעים (immutable). אחרי שערכו של עצם כזה נקבע, הוא אינו משתנה יותר. ניתן לשנות התייחסות למחרוזת כך שתתייחס למחרוזת אחרת, אבל לא ניתן לשנות את המחרוזות עצמן.

ניתן שוויון בין עצמים מקובעים בצורה יעילה על ידי שמירת עצם אחד לכל היותר מכל ערך אפשרי. יוצרים מבנה נתונים שמאפשר חיפוש, כמו עץ חיפוש או טבלת גיבוב. השירות היחיד שמבנה הנתונים הזה מספק הוא הכנסת עצם לתוכו. השירות מחזיר התייחסות לעצם. אם בזמן הקריאה לשירות אין במבנה הנתונים עצם עם ערך זה, השירות מכניס את העצם למבנה הנתונים ומחזיר התייחסות לעצם שהוכנס. אם יש במבנה עצם עם ערך זה, השירות מחזיר התייחסות לעצם שבמבנה, ולא מכניס למבנה את העצם שהועבר. בין התייחסויות שהוחזרו מהשירות הזה מתקיים התנאי שהתייחסויות שוות אם העצמים שהועברו לשירות זהים. הכנסת העצמים למבנה עשויה להיות יקרה, אבל לאחר מכן בדיקת זהות היא זולה. ג'אווה מספקת מבנה כזה דרך השירות intern על מחרוזות:

```
s = s.intern();
t = t.intern();
if ( s == t ) ... // returns true
```

תוויות (labels)

פסוקי break ו-continue יכולים לעצור או להמשיך לא רק את הלולאה הפנימית ביותר, אלא גם לולאות חיצונית, בעזרת שימוש בתווית לסימון הלולאה או גוש הפסוקים שהפסוק מתייחס אליהם. השגרה הבאה, למשל, משתמשת ב-break על מנת לעצור חיפוש,

```
boolean containsZeros(double[][] A, int n) {
    boolean found = false;

    outer: for (int i=0; i < n; i++) {
        for (int j=0; j < n; j++) {
            if (A[i][j] == 0.0) {
                found = true;
                break outer;
            }
        }
    }
    return found;
}
```

למעשה, אפשר היה לחזור מהשגרה כאשר מוצאים 0, אבל אם השגרה הייתה צריכה לעשות משהו אם הערך שנמצא, אז פסוק ה-break היה באמת נחוץ.