מחלקות

המחלקות בשפת ++C הן שילוב והרחבה של המבנים הקיימים ב-C וב-Java.

בשפת C, הדרך המקובלת ליצור מבנים מורכבים היא להגדיר struct. המשמעות של struct היא פשוט אוסף של משתנים שנמצאים ברצף בזיכרון. אפשר גם להגדיר משתנים מורכבים יותר ע"י struct בתוך struct וכו'. אבל ב-struct אין שיטות - אם רוצים לעשות איתו משהו צריך לכתוב שיטות חיצוניות.

בשפת Java יש מחלקות (class) שאפשר לשים בהן גם משתנים וגם שיטות - זה אמור להפוך את הקוד לקריא יותר כי כל המידע והפעולות הדרושות למימוש העצם נמצאים במקום אחד. אבל, בניגוד למבנים של סי, בג'אבה המשתנים לא תמיד נמצאים ברצף בזיכרון. לדוגמה, נניח שאנחנו מגדירים מבנה של "נקודה" ובו שני מספרים שלמים, ואז מגדירים מבנה של "משולש" ובו שלוש נקודות. בסי, כל משולש כולל פשוט שישה מספרים הנמצאים ברצף בזיכרון, ותופס 24 בתים בדיוק. בג'אבה, כל משולש כולל שלושה **מצביעים** (pointers), כל אחד מהם מצביע לנקודה שבה יש שני מספרים. כלומר המספרים לא נמצאים ברצף בזיכרון. למה זה משנה?

- בלי מצביעים, התוכנה תופסת פחות זיכרון ודורשת פחות זמן כשניגשים למשתנים.
- כשכל המבנה נמצא באותו מקום בזיכרון, ניהול הזיכרון מהיר יותר. בפרט, כשמשתמשים בזכרון מטמון (cache), זה מאד יעיל שכל המבנה נטען בבת-אחת לאותו בלוק בזיכרון.

שפת ++C משלבת את היתרונות של שתי השפות:

- יש בה struct כמו ב-C, אבל אפשר לשים בו גם שיטות.
- יש בה class כמו ב-Java, אבל המשתנים נשמרים ברצף בזיכרון ולא ע"י פוינטרים (נראה בהמשך).

למעשה, ב-++C, ההבדל היחיד בין struct לבין class הוא בהרשאות הגישה: בראשון, כברירת מחדל, הרשאת הגישה היא public; בשני, כברירת מחדל, הרשאת הגישה היא private. מכאן שההגדרות הבאות שקולות לחלוטין (ראו דוגמה בתיקיה 1):

```
struct X {
    private:
          ...
}
===
class X {
          ...
}
```

וכן ההגדרות הבאות שקולות לחלוטין:

```
struct X {
          ...
```

```
}
===
class X {
     public:
          ...
}
```

דרכים למימוש שיטות

יש שתי דרכים לממש שיטות של מחלקות ב-++C.

- דרך אחת היא לממש אותן ישירות בתוך המחלקה (בדומה לJava). זה נקרא מימוש inline ודומה לפונקציות inline שראינו בשיעור הקודם, רק שכאן לא צריך לכתוב את המילה inline.
- דרך שניה היא לשים בתוך המחלקה רק **הצהרה** של השיטה, בלי גוף. את המימוש עצמו שמים מחוץ למחלקה (באותו קובץ או בקובץ אחר). ראו דוגמה בתיקיה 2.

המצביע this

במימוש של שיטה, המילה השמורה this מכילה מצביע לעצם הנוכחי. ניתן להשתמש בה כמו שמשתמשים במצביעים, למשל:

```
Point::Point(int x, int y) {
      this->x = x;
      this->y = y;
}
```

(זה דומה לג'אבה פרט לכך שמשתמשים בחץ במקום בנקודה).

שדות סטטיים

ניתן להגדיר שדות ושיטות סטטיים (שייכים לכל המחלקה ולא לעצם מסויים) בעזרת המילה השמורה static. כדי לגשת אליהם מבחוץ, מקדימים להם את שם המחלקה ופעמיים נקודתיים, למשל Point::MAXX. אם מגדירים שדה סטטי שהוא גם קבוע (const), ניתן לאתחל אותו בהגדרת המחלקה; אחרת, יש לאתחל אותו מבחוץ (ראו דוגמה בתיקיה 3).

קבצי כותרת למחלקות

מקובל (אם כי לא חובה) להגדיר כל מחלקה בשני קבצים (ראו דוגמה בתיקיה 4):

- קובץ הכותרת - בדרך-כלל עם סיומת hpp או h - כולל את הגדרת המחלקה, השדות והשיטות שלה - אבל בלי מימוש השיטות.
- קובץ התוכן - בדרך-כלל עם סיומת cpp - כולל את המימוש של השיטות.

עקרונית, ניתן לממש את כל השיטות בקובץ הכותרת (כמו בג'אבה), אבל זה לא כדאי. מדוע? כמה סיבות:

1. הנדסת תוכנה. אם ניתן את המחלקה שלנו למישהו אחר, הוא ירצה לראות איזה שיטות יש בה, אבל לא יעניין אותו לדעת איך בדיוק מימשנו אותן. לכן הוא יסתכל בקובץ הכותרת, ועדיף שהקובץ הזה יהיה קטן ופשוט ככל האפשר.
2. זמן קומפילציה. במערכות תוכנה מורכבות, קומפילציה לוקחת הרבה זמן. בכל פעם שמשנים קובץ, צריך לקמפל מחדש את כל הקבצים שתלויים בו. כששמים את המימוש בקובץ נפרד, שינוי במימוש לא דורש קימפול מחדש של קוד שמשתמש בקובץ הכותרת. כדוגמה ראו את ה-`Makefile` בתיקיה 4.

השימוש בקבצי-כותרת פותר את הבעיה באופן חלקי בלבד. מדוע? כי המשתנים הפרטיים - שהם חלק מהמימוש - עדיין נמצאים בקובץ הכותרת. המשתמשים של המחלקה שלנו רואים אותם, ואם נשנה אותם נצטרך לבנות את כל הפרוייקט מחדש. יש לזה פתרונות - נראה בהמשך.

מקורות

- מצגות של אופיר פלא ומירי בן-ניסן.

סיכם: אראל סגל-הלוי.