שיטות קבועות - const methods

המילה const קיימת גם בשפת סי. אם שמים אותה מייד לפני שם של משתנה - המשתנה יהיה קבוע, והקומפיילר לא ייתן לנו לשנות אותו. עד כאן זה פשוט.

כשיש פוינטרים זה קצת יותר מסובך - צריך לשים לב מי הקבוע - הפוינטר או הדבר שהוא מצביע עליו?

```
int *const p1 = &i; // a const pointer to an un-const variable
p1++;     // compiler error
(*p1)++; // ok
const int* p2 = &b; // an un-const pointer to a const variable
p2++;      // ok
(*p2)++;  // compiler error
const int * const p3 = &b; // a const pointer to a const variable
```

כשיש מחלקות ועצמים, המצב מסתבך עוד יותר. אפשר לשים את המילה const בכותרת של שיטה, אחרי הסוגריים. המשמעות היא, שבתוך השיטה הזאת, המשתנה this יהיה מצביע לעצם קבוע. במילים אחרות: השיטה לא תוכל לשנות את השדות של העצם.

למה זה חשוב?

- זה עוזר לאתר באגים ותקלות. אם שיטה מסויימת מוגדרת כ-const, אפשר להיות בטוחים שהיא לא משנה את העצם, ולכן אם העצם משתנה כנראה התקלה במקום אחר.
- אם בתוכנית הראשית מגדירים עצם כ-const, אפשר לקרוא על העצם הזה רק לשיטות שהוגדרו כ-const.
- זה מאפשר לנו לקבל אזהרות על תופעות-לוואי לא רצויות. למשל, נניח שאנחנו כותבים פונקציה print שמדפיסה את אחד השדות של העצם שלנו. אבל תוך כדי ההדפסה, אנחנו קוראים לפונקציית-ספריה שמשנה את העצם (יש פונקציות כאלו בספריה התקנית! למשל אופרטור סוגריים מרובעים של map עלול לשנות את העצם!). אם נגדיר את השיטה כ-const, הקומפיילר יזהה את הבעיה ויזהיר אותנו.

למה זה קשה? כי כששיטה היא const, הקומפיילר לא ייתן לנו לקרוא מתוכה לשיטות אחרות שהן לא const! לכן, כשמגדירים שיטה כ-const עלולה להיווצר "תגובת שרשרת" שתדרוש מאיתנו הרבה שינויים בקוד. לכן עדיף מלכתחילה להגדיר כ-const כל שיטה שאנחנו יודעים שלא תצטרך לשנות את העצם.

תרגיל בית: קחו את התרגילים הקודמים שלכם, הוסיפו להם "const" במקומות הנכונים, וראו מה קורה..

קבועים והעמסה

כשמוסיפים את המילה const לשיטה, היא הופכת לחלק מה"חתימה" של השיטה. מכאן שאפשר ליצור שתי שיטות שונות עם אותו שם ואותם פרמטרים - אחת עם const ואחת בלי. הקומפיילר ישתמש בשיטה הנכונה לפי ההקשר - אם משתמשים בשיטה כ-lvalue הוא יקרא לשיטה בלי ה-const, ואם משתמשים בשיטה כ-rvalue הוא יקרא לשיטה עם ה-const.

מתי זה שימושי? למשל, כשמגדירים מבנה-נתונים של וקטור. למבנה יש שיטה `get(i)` שמחזירה את האלמנט במקום `i`. מקובל ליצור שתי שיטות עם **מימוש זהה**:

- אחת מיועדת לקריאה - היא מוגדרת כ-`const` ומחזירה `const&`.
- השניה מיועדת לכתיבה - היא מוגדרת בלי `const` ומחזירה `&`.

הקומפיילר יחליט לאיזו שיטה לקרוא, לפי סוג המשתנה: אם המשתנה הוא `const` הוא יקרא לשיטה המיועדת לקריאה בלבד; אחרת הוא יקרא לשיטה המיועדת לכתיבה.

שדה `mutable`

לפעמים רוצים לשמור בתוך עצם, שדה מסויים שהערך שלו לא משקף את המצב של העצם. לדוגמה, זה יכול להיות `cache` של תוצאת-ביניים של חישוב כלשהו, או מונה הסופר את מספר הגישות לפונקציה מסויימת. החישוב לא משנה את מצב העצם מבחינה לוגית, אבל הוא צריך לשמור את התוצאה בתוך העצם.

כדי שהקומפיילר יאפשר לנו לעשות זאת, נסמן את ה-`cache` ב-`mutable`.

שדה המסומן ב-`mutable` ניתן לשינוי גם אם העצם הוא `const`.

מקורות

- מצגות של אופיר פלא ומירי בן-ניסן.
- למה חשוב שהשיטות יהיו `const` - מפי מפתח בכיר בפייסבוק
  `https://youtu.be/lkgszkPnV8g`

סיכם: אראל סגל-הלוי.