מחלקות - בניה ופירוק

בשפת ++C יש עקרון חשוב:

- כל עצם חייב לעבור תהליך של בניה (construction) ברגע שהוא נוצר; הבניה מתבצעת ע"י בנאי.
- כל עצם חייב לעבור תהליך של פירוק (destruction) ברגע שהוא מפסיק להתקיים; הפירוק מתבצע ע"י מפרק.

בנאים - constructors

בנאי הוא שיטה ששמה כשם המחלקה, ואין לה ערך-חזרה. כמו כל שיטה אחרת, ניתן לממש אותה בקובץ הכותרת או בקובץ המימוש. ראו דוגמאות בתיקיה 4.

כמו כל פונקציה, גם בנאים אפשר להעמיס. כלומר, אפשר להגדיר כמה בנאים עם פרמטרים שונים, והקומפיילר יקרא לבנאי הנכון לפי השימוש.

איך קוראים לבנאי? תלוי:

- אם זה בנאי עם פרמטרים, פשוט שמים את הפרמטרים אחרי שם העצם, למשל Point; p(10,20) קורא לבנאי של Point המקבל שני מספרים שלמים.
- אם זה בנאי בלי פרמטרים, לא שמים שום דבר אחרי שם העצם, למשל Point p; (לא לשים סוגריים ריקים - זה יגרום לשגיאת קימפול כי הקומפיילר עלול לחשוב שאתם מנסים להגדיר פונקציה...)

שימו לב - בניגוד לג'אבה - לא צריך להשתמש ב-new! המשתנה נוצר "על המחסנית" ולא "על הערימה" (אם נשתמש ב-new, המשתנה ייווצר על הערימה).

בנאי ללא פרמטרים

יש כמה סוגי בנאים שיש להם תפקיד מיוחד ב-++C. אחד מהם הוא **בנאי ללא פרמטרים** (parameterless constructor). אם (ורק אם) לא מגדירים בנאי למחלקה - הקומפיילר אוטומטית יוצר עבורה בנאי כזה; הוא נקרא default parameterless constructor.
בהתאם לגישה של ++C "לא השתמשת - לא שילמת", בנאי ברירת-מחדל של מחלקה פשוטה לא עושה כלום - הוא **לא מאתחל את הזיכרון** (בניגוד לג'אבה). לכן, הערכים לא מוגדרים - ייתכן שיהיו שם ערכים מוזרים שבמקרה היו בזיכרון באותו זמן.

בנאים נוספים נראה בהמשך.

מפרקים - destructors

אנחנו מגיעים עכשיו לאחד ההבדלים העיקריים בין ++C לג'אבה. כזכור, ב-++C ניהול הזיכרון הוא באחריות המתכנת. בפרט, אם אנחנו יוצרים משתנים חדשים על הערימה, אנחנו חייבים לוודא שהם משוחררים כשאנחנו כבר לא צריכים אותם יותר. לשם כך, בכל מחלקה שמקצה זיכרון (או מבצעת פעולות אחרות הדורשות משאבי מערכת), חייבים לשים **מפרק - destructor.**

מפרק הוא שיטה בלי ערך מוחזר, ששמה מתחיל בגל (טילדה) ואחריו שם המחלקה; ראו דוגמה בתיקיה 5 (למה גל? כי זה האופרטור המציין "not" של סיביות. למשל: 0~=-1).

חידה: האם אפשר להגדיר מפרק עם פרמטריס? האם אפשר לבצע העמסה (*overload*) למפרק? מדוע?

המתכנת אף-פעם לא צריך לקרוא למפרק באופן ידני; זוהי האחריות של הקומפיילר להכניס קריאה למפרק ברגע שהעצם מפסיק להתקיים. מתי זה קורה?

- אם העצם נוצר על המחסנית - העצם מפסיק להתקיים כשהוא יוצא מה-scope (יוצאים מהבלוק המוקף בסוגריים מסולסלים המכיל את העצם).
- אם העצם נוצר על הערימה בעזרת new - העצם מפסיק להתקיים כשמוחקים אותו בעזרת delete.

מה קורה כששוכחים לשים מפרק? המשאבים לא ישתחררו, וכתוצאה מכך תהיה דליפת זיכרון; ראו הדגמה בתיקיה 5.

האופרטורים new, delete

האופרטור new מבצע, כברירת מחדל, את הדברים הבאים:

- הקצאת זיכרון עבור עצם חדש מהמחלקה;
- קריאה לבנאי המתאים של המחלקה (בהתאם לפרמטרים שהועברו).

האופרטור delete מבצע, כברירת מחדל, את הדברים הבאים:

- קריאה למפרק של המחלקה (יש רק אחד - אין פרמטרים);
- שיחרור הזיכרון שהוקצה עבור העצם.

האופרטור []new מבצע, כברירת מחדל, את הדברים הבאים:

- הקצאת זיכרון עבור מערך של עצמים מהמחלקה;
- קריאה לבנאי ברירת-המחדל של כל אחד מהעצמים במערך.

האופרטור []delete מבצע, כברירת מחדל, את הדברים הבאים:

- קריאה למפרק של כל אחד מהעצמים במערך.
- שיחרור הזיכרון שהוקצה עבור המערך.

כשמאתחלים מערך ע"י new[], חייבים לשחרר אותו ע"י delete[]. כאן הקומפיילר לא יציל אתכם משגיאה - אם תשתמשו ב-delete במקום delete[], הקוד יתקמפל, אבל הקומפיילר יקרא רק למפרק אחד (של האיבר הראשון במערך). כתוצאה מכך תהיה לכם דליפת זיכרון, או אפילו שגיאת ריצה (ראו תיקיה 6).

מקורות

- מצגות של אופיר פלא ומירי בן-ניסן.

סיכם: אראל סגל-הלוי.