# עץ מחווט

ממ"ן 16

מגישים:

|  |  |
| --- | --- |
| ת.ז | שם |
| 315532010 | בר שביט |
|  | סבטה סילה |

# תיאור הבעיה

עץ חיפוש בינארי רגיל בן n צמתים מכיל n + 1 מצביעי Left ו-Right שערכם NIL. במצב זה זיכרון רב "מתבזבז".

הגישה הכללית של התוכנית לפתרון הבעיה

נחליף את מצביעי ה-Left המכילים NIL של הצומת z במצביע ל-.

באופן דומה, נחליף את מצביעי ה-Right המכילים NIL של הצומת z במצביע ל-TREE-SUCCESSOR(z).

כך נבנה עץ מחוות.

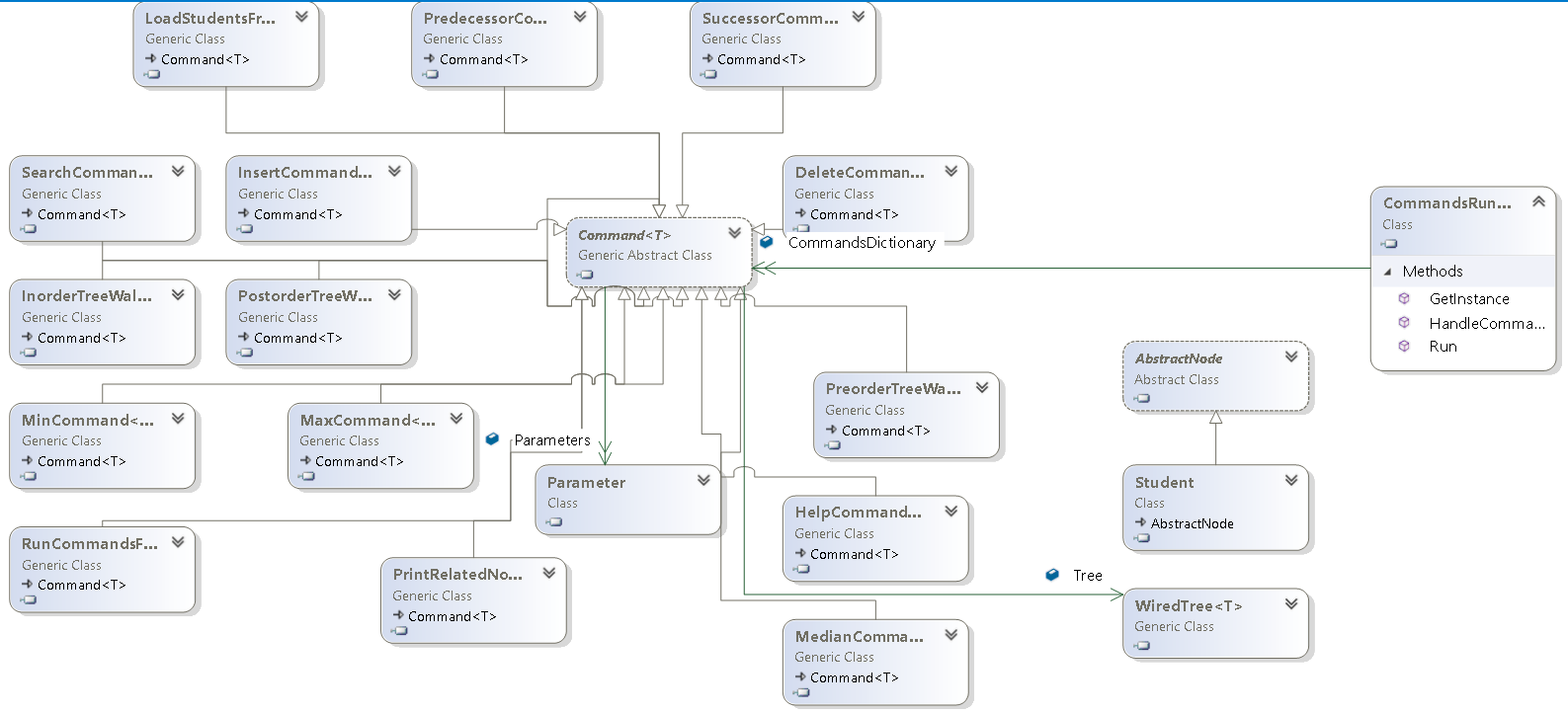
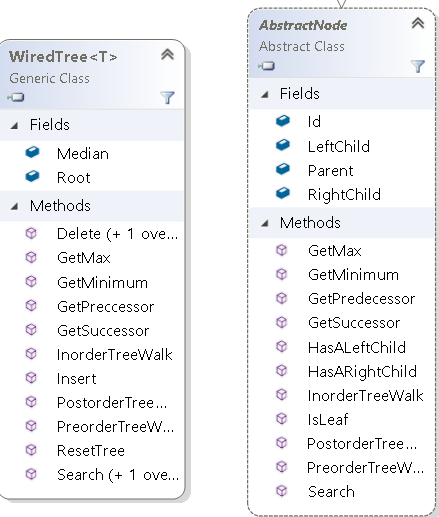
# מבנה הנתונים

התוכנית מתבססת על עץ חיפוש בינארי, אליו *הוספנו את לוגיקת החוטים.*

*החוטים מתוחזקים בעת הכנסה ומחיקה מהעץ.*

*על מנת להבדיל בין מצביעים חוטים למצביעים לבנים אמיתיים, מימשנו 2 פונקציות עבור כל צומת – HasARightChild, HasALeftChild הבודקות האם לצומת יש בן אמיתי. במקרה שהמצביע הוא לא NIL, אנו בודקים אם האבא של הצומת הבן הוא הצומת הנוכחית, אם כן – זה הוא בן אמיתי.*

# תיאור כללי של פונקציות

*  (קובץ ה-UML נמצא ביחד עם הפרויקט).*

***הפרויקט מורכב משני חלקים:***

1. *העץ עצמו – כל צומת ממומש על ידי AbstractNode המייחצן את הפעולות הבסיסיות שמתבצעות על צמתים. במקרה זה, יצרנו Student היורש מ-AbstractNode ומוסיף מעליו את פרטי הסטודנט.  
   המחלקה WiredTree מחזיקה מצביע לשורש ולחציון.  
   המחלקה אחראית ליחצן למשתמשים את כל פונקציונליות העץ ותחזוק המצביע לחציון.*
2. *הרצת פקודות – השתמשנו ב-Command Design Pattern.  
   יצרנו מחלקה בסיסית של פקודה בשם Command המחזיקה שם פקודה, תיאור כללי, אילו פרמטרים היא מקבלת ומצביע לעץ עליו היא עובדת.  
   כאשר הלקוח רוצה להשתמש בפקודות, הוא ניגש למחלקת CommandsRunner (Singleton) המחזיקה מילון המכיל מופעים של כל הפקודות ומאפשרת הרצת CLI.*

# פקודות

*לאחר הרצת התוכנית, ניתן לכתוב את הפקודה help ולקבל רשימה מלאה ומפורטת של כל הפעולות הנתמכות (כולל הרצת פקודות מקובץ) כולל הפרמטרים הנדרשים.*

***נדרש .Net Framework 4.6.1 ומעלה על מנת להריץ את הפרויקט.***