אוניברסיטת בן גוריון

הפקולטה למדעי המחשב

מבוא לפירוק והרכבה של תכניות – פרויקט סיום

*מטרת ונושא הפרוייקט: הדגמת אלגוריתמי code-motion וrefactoring שנלמדו במהלך הסמסטר.*

מרצה הקורס: ד"ר רן אטינגר

מגישים: טל יצחק (204260533)

מאי נגלר (200935963)

כלל הקוד בפרויקט נכתב בJava, וזמין ב-GitHub:

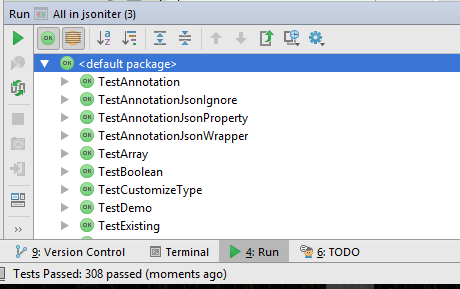
github.com/talitz/Introduction-to-Code-Motion-Refactoring-Project

סמסטר ב', תשע"ז, 2017

רקע

הפרויקט מממש Parser גמיש ומהיר ל-JSON, הזמין בשפות Java ו-Go. הפרויקט נלקח מהrepository הזמין לצפייה בקישור הבא: <https://github.com/json-iterator/java>.

הפרויקט נכתב עם סביבת טסטים ב- JUnit, להלן המחשה של מעבר כלל הטסטים:



בפרויקט נציג דוגמאות מקטעי קוד נבחרים ונריץ את האלגוריתמים שלמדנו בכיתה לrefactoring & code motion.

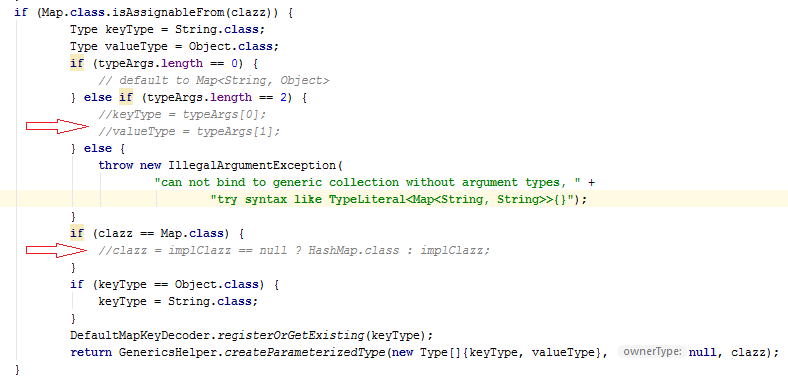
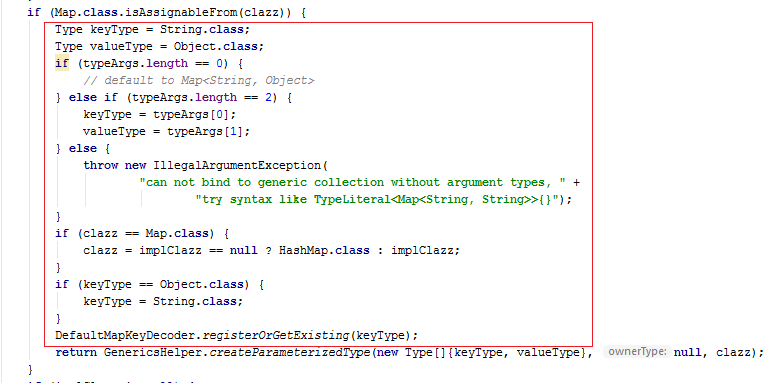
הערה: כאשר נבחר קטעי קוד ספציפיים עליהם נרצה לבצע refactoring, נשנה מעט את הקוד המקורי כדי לוודא שאכן הטסטים הנ"ל נכשלים, כלומר אכן בודקים את נכונותו של אותו קטע הקוד הספציפי.

בסוף הrefactoring נריץ שוב את כלל הטסטים כדי לוודא שהשינוי משמר את הנכונות.

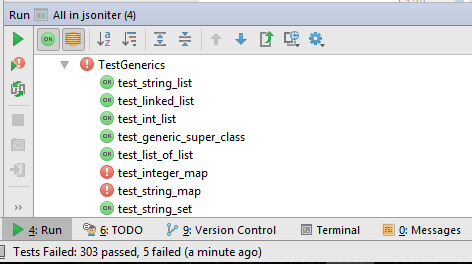
דוגמא 1: PDG-Based Sliding

נתבונן בחתיכת הקוד הבאה:

נבצע sanity checking, כלומר ווידוא שאכן כלל הטסטים שקיימים בפרויקט בודקים את נכונות התכנית הנ"ל. נשנה מעט את התכנית (שמנו חלק מהשורות בקוד בהערות):



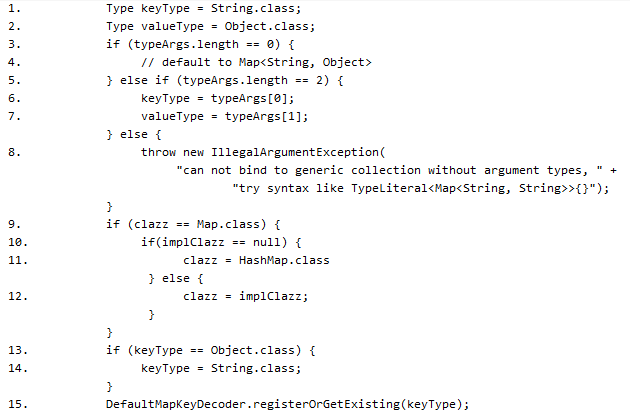
כעת, נצפה לראות שחלק מהבדיקות נכשלות. נריץ את כלל הבדיקות:



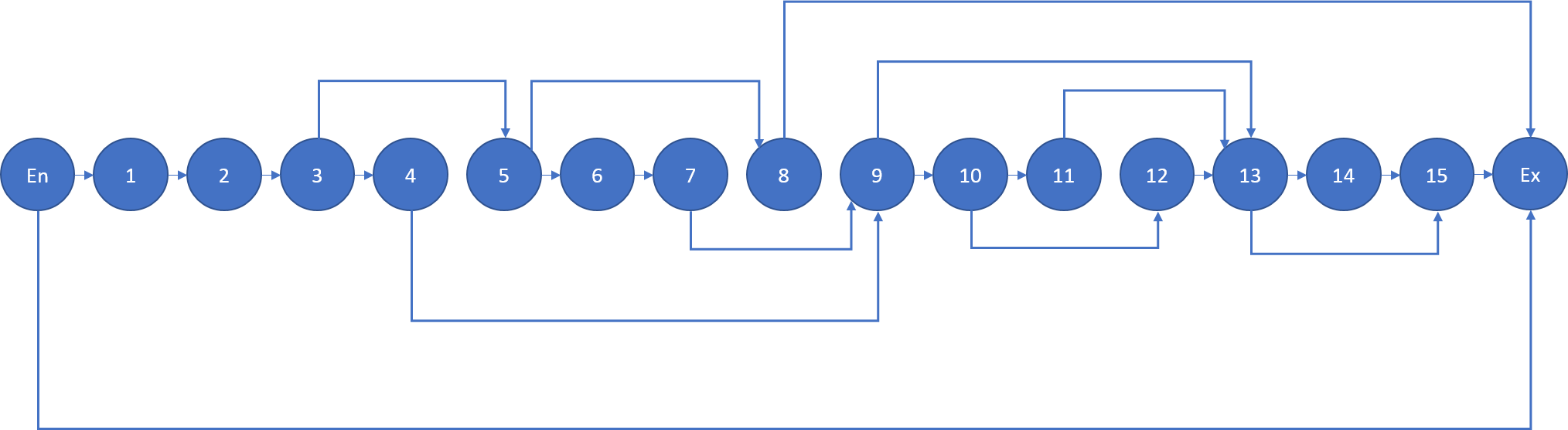
ניתן לראות כי 5 בדיקות אכן נכשלו – משמע כי אכן שינוי בקוד הנ"ל יכול להביא לכישלון הבדיקות, ז"א – כישלון התכנית.

כעת, נרצה לבצע PDG-Base Sliding כאשר קבוצת המשתנים שנבחר תהיה .

לשם נוחות נמספר מחדש את כלל שורות הקוד בקטע הנבחר:



תחילה נבחר לבנות את הCFG עבור קטע הקוד הנבחר:



כבסיס לאלגוריתם הSliding על קטע הקוד המורכב מהפקודות 1-15 שלעיל, נקבל את הייצוג הבא בגרף הPDG, כאשר קבוצת הקודקודים N בה נעשה שימוש באלגוריתם היא:

וקבוצת הקשתות E הכוללת תלויות מסוג control (הקבוצה Ec) ותלויות data (הקבוצה Ed) מסוג flow או anti כדלהלן:

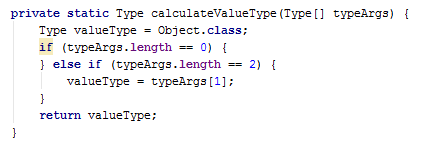
|  |  |  |
| --- | --- | --- |
| Vars | Kind | Edge |
| - | Control | (Entry,1) |
| - | Control | (Entry,2) |
| - | Control | (Entry,3) |
| - | Control | (3,4) |
| - | Control | (3,5) |
| - | Control | (3,9) |
| - | Control | (3,13) |
| - | Control | (3,15) |
| - | Control | (3,16) |
| - | Control | (5,6) |
| - | Control | (5,7) |
| - | Control | (5,8) |
| - | Control | (5,13) |
| - | Control | (5,15) |
| - | Control | (5,16) |
| - | Control | (9,10) |
| - | Control | (10,11) |
| - | Control | (10,12) |
| - | Control | (13,14) |
| {clazz} | Flow | (Entry,9) |
| {implClazz} | Flow | (Entry,10) |
| {implClazz} | Flow | (Entry,12) |
| {keyType} | Flow | (1,13) |
| {keyType} | Flow | (1,15) |
| {keyType} | Flow | (6,13) |
| {keyType} | Flow | (6,15) |
| {clazz} | Anti | (9,11) |
| {clazz} | Anti | (9,12) |
| {keyType} | Anti | (13,14) |
| {keyType} | Flow | (14,15) |
| {keyType} | Flow | (1,Exit) |
| {keyType} | Flow | (6,Exit) |
| {keyType} | Flow | (14,Exit) |
| {valueType} | Flow | (2,Exit) |
| {valueType} | Flow | (7,Exit) |
| {clazz} | Flow | (11,Exit) |
| {clazz} | Flow | (12,Exit) |
| {IllegalArgumentException} | Flow | (8,Exit) |

כעת נפעיל את אלגוריתם הSliding עבור .

תוצאת האלגוריתם:

1. הקוד המחולץ (הSlice, תת הקבוצה Nv) יהיה:
2. הקוד המשלים (הCo-Slice, תת הקבוצה Ncov) יהיה:

נבצע Extract Method לSlice שהתקבל:



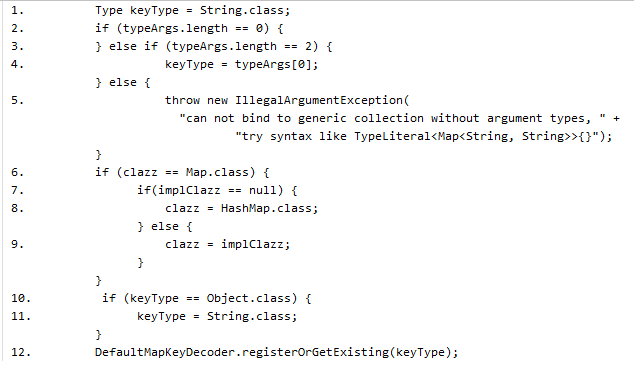
נתבונן כעת בקוד, רגע לפני ביצוע Extract Method ל-Co-Slice:



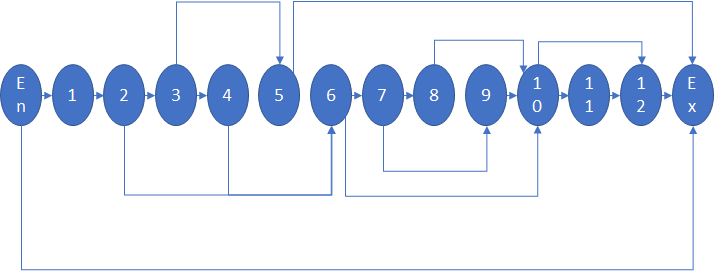
נשים לב שמתבצע חישוב בCo-Slice עבור המשתנים clazz ו-keyType, כלומר כדי לבצע Extract Method ל-Co-Slice, נצטרך לאחסן את הנתונים החוזרים מהפונקציה באובייקט שיכיל 2 שדות עבור ערכים אלו (זה מה שמוצע ע"י IntelliJ כשמבצעים Extract Method במצב שכזה).

אולם, אנו מזהים כאן פוטנציאל להריץ שוב את אלגוריתם ה-PDG Based Sliding, על קבוצת המשתנים .

נמספר את שורות הקוד מחדש (עבור קטע ספציפי זה):



נבנה את ה-CFG עבור קטע הקוד הנ"ל:



כבסיס לאלגוריתם הSliding על קטע הקוד המורכב מהפקודות 1-12 שלעיל, נקבל את הייצוג הבא בגרף הPDG, כאשר קבוצת הקודקודים N אשר בה נעשה שימוש באלגוריתם היא:

וקבוצת הקשתות E הכוללת תלויות מסוג control (הקבוצה Ec) ותלויות data (הקבוצה Ed) מסוג flow או anti כדלהלן:

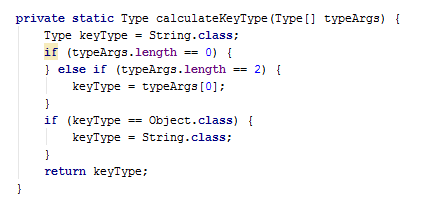
|  |  |  |
| --- | --- | --- |
| Vars | Kind | Edge |
| - | Control | (Entry,1) |
| - | Control | (Entry,2) |
| - | Control | (2,3) |
| - | Control | (2,6) |
| - | Control | (2,10) |
| - | Control | (2,12) |
| - | Control | (3,4) |
| - | Control | (3,5) |
| - | Control | (3,6) |
| - | Control | (3,10) |
| - | Control | (3,12) |
| - | Control | (6,7) |
| - | Control | (7,8) |
| - | Control | (7,9) |
| - | Control | (10,11) |
| {clazz} | Flow | (Entry,6) |
| {implClazz} | Flow | (Entry,7) |
| {implClazz} | Flow | (Entry,9) |
| {keyType} | Flow | (1,10) |
| {keyType} | Flow | (1,12) |
| {keyType} | Flow | (1,Exit) |
| {keyType} | Flow | (4,10) |
| {keyType} | Flow | (4,12) |
| {keyType} | Flow | (4,Exit) |
| {clazz} | Anti | (6,8) |
| {clazz} | Anti | (6,9) |
| {IllegalArgumentException} | Flow | (5,Exit) |
| {clazz} | Flow | (8,Exit) |
| {clazz} | Flow | (9,Exit) |
| {keyType} | Anti | (10,11) |
| {keyType} | Flow | (11,12) |
| {keyType} | Flow | (11,Exit) |

כעת נפעיל סליידינג עבור .

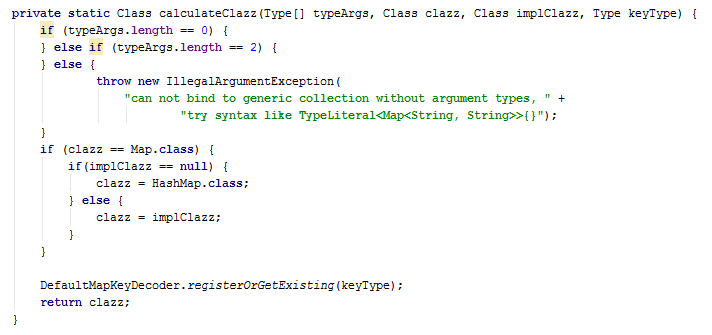
תוצאת הסליידינג:

1. הקוד המחולץ (הסלייס, תת הקבוצה Nv) יהיה:
2. הקוד המשלים את הקו-סלייס יהיה (Ncov):

כעת נוכל לבצע Extract Method לslice:



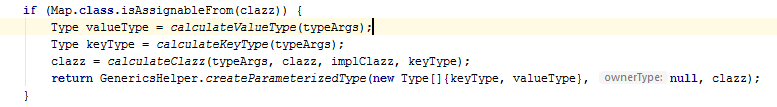
ולco-slice:



הערה: נשים לב כי זריקת ה-Exception נעשית ב-co-slice ולא ב- slice. לאחר חשיבה מעמיקה בנושא מצאנו כי נכון להתייחס למקרה זה באופן פרטני כך שקיימת קשת flow מקודקוד 5 (הקודקוד בו נזרק הException) ל-Exit, כלומר הוא "חי" בסוף התכנית.

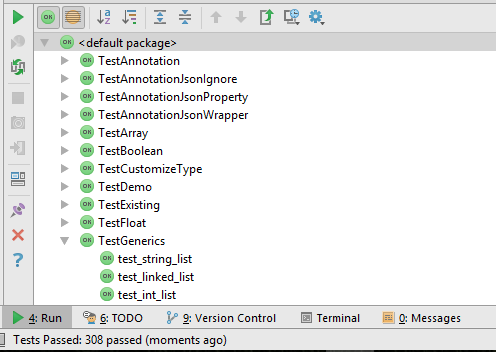
במקרה כללי בו נזרקים Exceptions, ניתן לבצע עיבוד מקדים של התכנית כך שתחילה ייבדקו כלל התנאים הנדרשים לזריקת הException, ואם אף אחד מהם לא מתקיים – התכנית רצה כרגיל, ועליה מתבצע אלגוריתם הPDG-Based Sliding (כלומר בלי קודקודים בהם נזרקים Exceptions).

ובסה"כ נקבל את הקוד הבא:



נוודא כעת שכלל הבדיקות עדיין עוברות (קליק ימני על הפרוייקט –

run 'All Tests'):

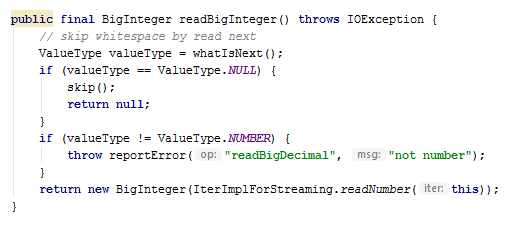


כנדרש.

כעת, ניתן להגיד בוודאות כי הקוד יותר ברור לקריאה ושימוש. בנוסף, עקב פיצול הקוד הרווחנו גמישות בקוד הנ"ל וכעת ניתן לבצע שינויים בקלות ולדאוג לתחזוקה עבור חלקים ספציפיים בקוד – מבלי שינוי קטע הקוד כולו. כמו כן שימוש חוזר בחלקים ספציפיים מהקוד ניתנים כעת לביצוע מבלי חזרה כוללת על קטע הקוד כולו. השגנו את מטרותינו.

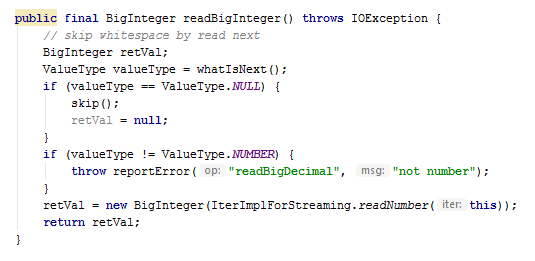
דוגמא 2: Separate Query From Method

נתבונן בקטע קוד הבא מהמחלקה JsonIterator:

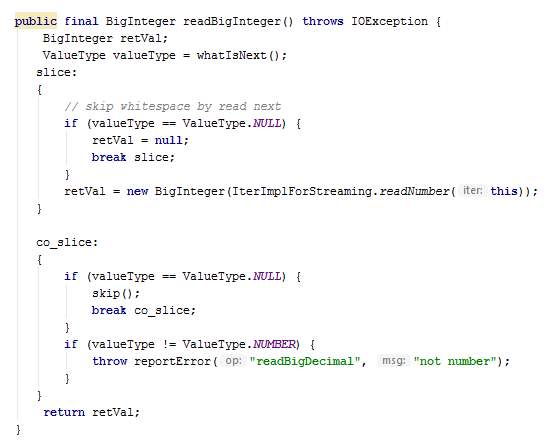


נעבוד לפי השיטה שהוצגה בהרצאות: :"Mechanics for SQfM"

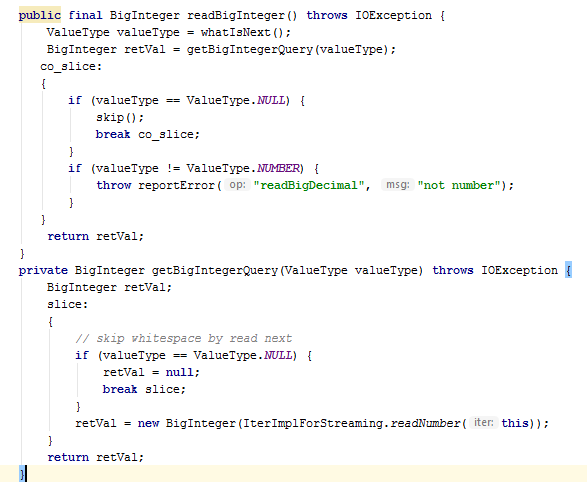
שלב 1: נכין את השיטה לSliding ע"י המרת הקוד הנכון כך שיהיה רק ערך מוחזר יחיד (הוספת ערך מוחזר ושינוי בהתאם למהלך התכנית):



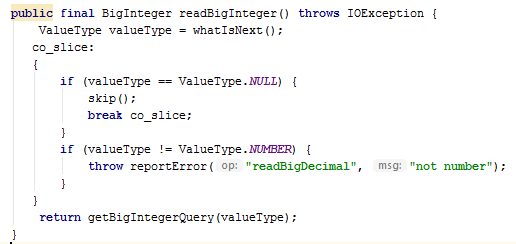
שלב 2: נבצע Sliding על הערך המוחזר retVal. נשים לב שהSlice הוא בעצם הQuery וה-Co-Slice הוא ה-Modifer:



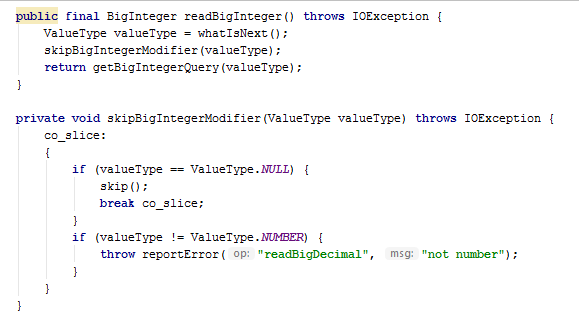
שלב 3: נבצע Extract Method ל-Query:



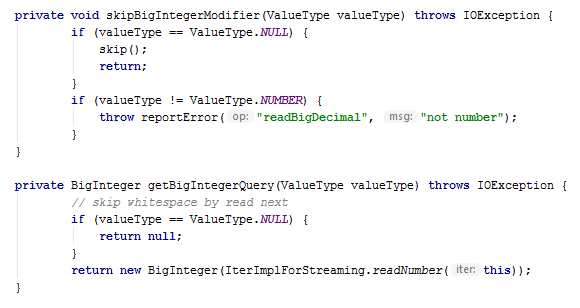
שלב 4: נבצע Inline Temp על הQuery:



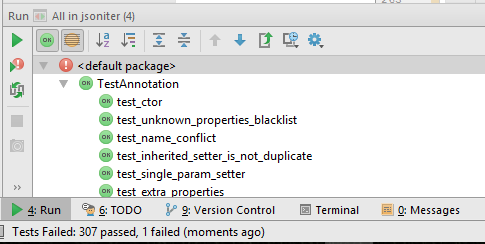
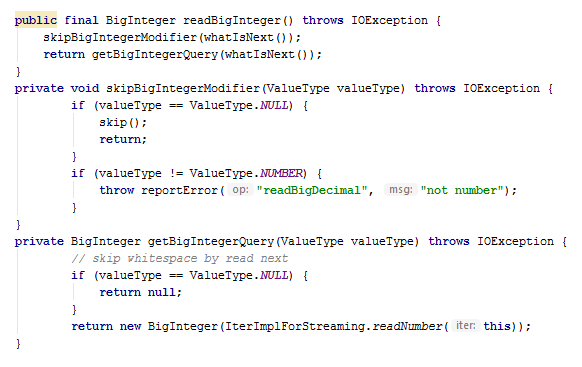
שלב 5: נבצע Extract Method עבור הModifier:



שלב 6: נבצע Undo לשלב ה-1 (החזרת הreturnים, מחיקת הlabelים):

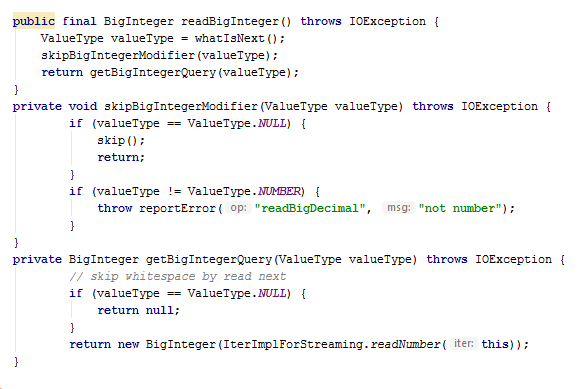


שלב 7: ביצוע Inline Method על הגרסה שלאחר הrefactoring: נשים לב שלא ניתן לבצע את הInline Method הבא (ואכן, טסט 1 נופל כאשר אנו מבצעים זאת):

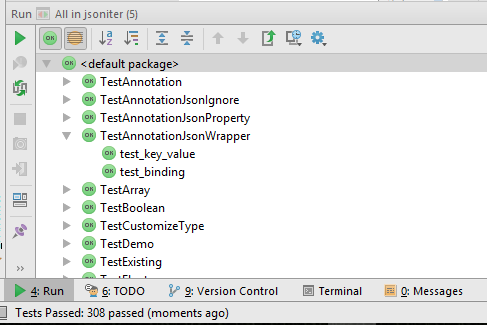


וזאת משום שהפונקציה whatIsNext() מבצעת side-effect. כלומר הInline Method היחיד שנוכל פוטנציאלית לבצע הוא לQuery שלנו, אבל לא קיים כזה.

לכן הגרסה הסופית תהיה:



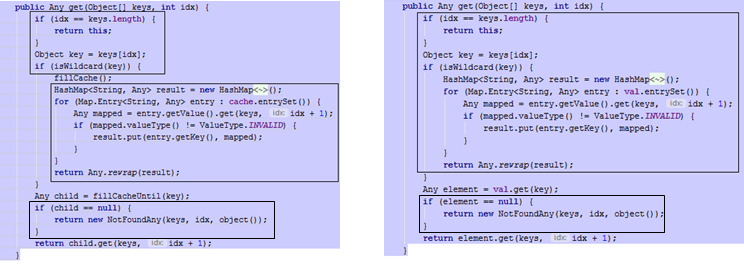
נציין שלאחר כל שלב ביצענו Compile & Test ווידאנו שכלל הבדיקות עוברות:



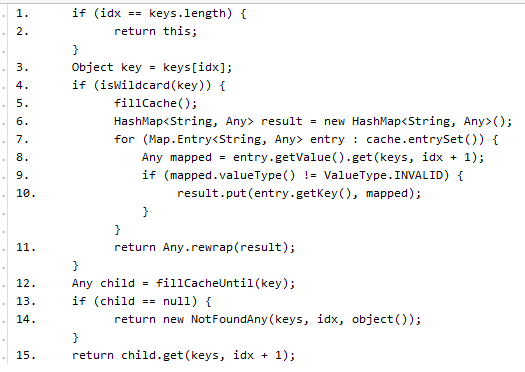
ובזאת סיימנו את כלל השלבים.

דוגמא 3: Slide-Based Bucketing

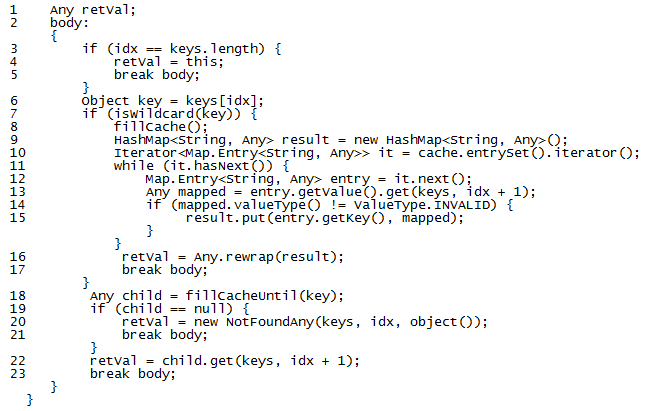
נתבונן בType-3 Clones הבאים, מהמחלקות ObjectAny, MapWrapperAny:

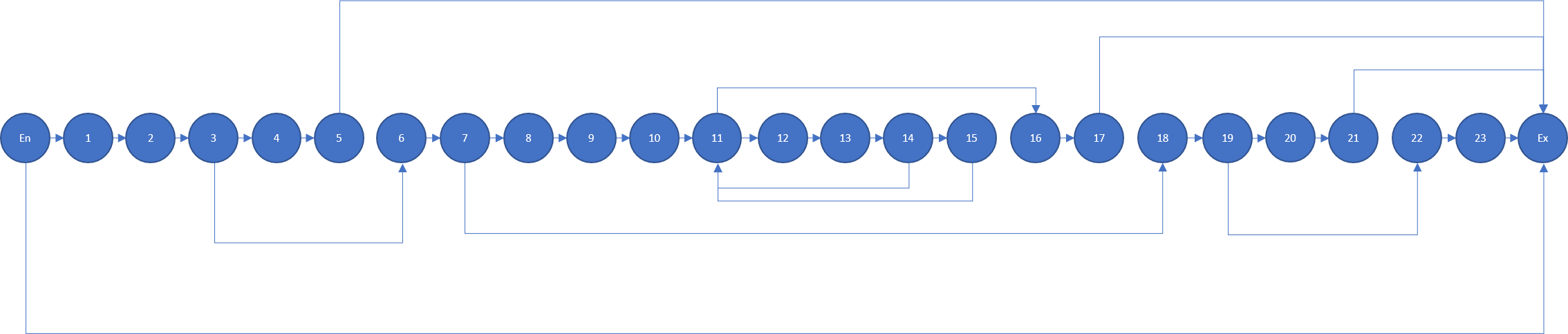


נתבונן בקטע הקוד השמאלי, נרצה לסמן את קטע הקוד המוקף בשחור כקבוצה M על מנת לבצע עליו Extract Method, נרצה להשתמש בSlide-Based Bucketing. נמספר את כלל השורות בקוד:



נבצע עיבוד מקדים לקוד הנ"ל כפי שנלמד בהרצאות:  
תחילה נהפוך את לולאת הforEach ללואת while בעזרת יצירת Iterator.  
לאחר מכן, נרצה לקבל קטע קוד בעל return יחיד, לכן ניצור משתנה ערך מוחזר (retVal) ונעטוף את קטע הקוד בbody – ועבור כל return בקטע הקוד המקורי – נבצע השמה לערך retVal, יציאה מקטע הbody והחזר הערך המתאים.  
לאחר העיבוד, זהו קטע הקוד שנקבל (כולל מס' השורות):



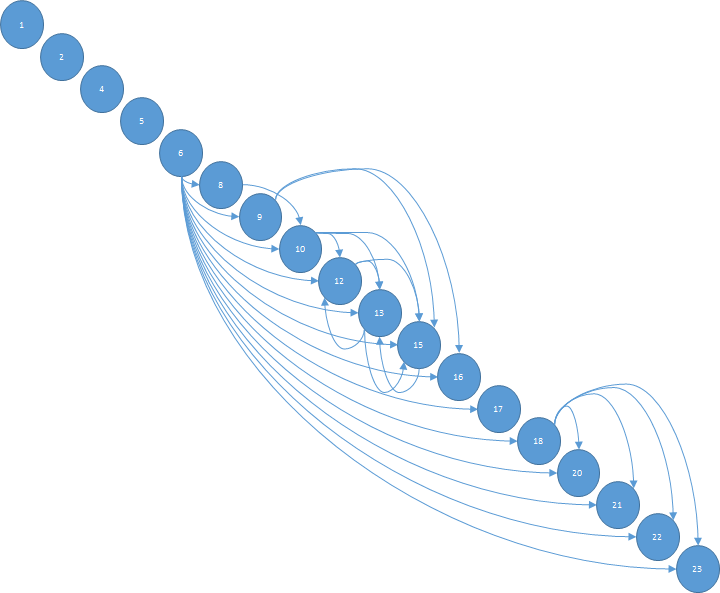
כעת נבנה את הCFG עבור קטע הקוד הנ"ל:

לקראת ביצוע האלגוריתם עבור פקודות מסומנות בקטע הקוד הכולל את הפקודות 1 עד 23, נבנה את הPDG בהתאם:

|  |  |  |
| --- | --- | --- |
| Vars | Kind | Edge |
| - | Control | (Entry,1) |
| - | Control | (Entry,2) |
| - | Control | (Entry,3) |
| - | Control | (3,4) |
| - | Control | (3,5) |
| - | Control | (3,6) |
| - | Control | (3,7) |
| - | Control | (7,8) |
| - | Control | (7,9) |
| - | Control | (7,10) |
| - | Control | (7,11) |
| - | Control | (7,16) |
| - | Control | (7,17) |
| - | Control | (7,18) |
| - | Control | (7,19) |
| - | Control | (11,12) |
| - | Control | (11,13) |
| - | Control | (11,14) |
| - | Control | (14,15) |
| - | Control | (19,20) |
| - | Control | (19,21) |
| - | Control | (19,22) |
| - | Control | (19,23) |
| {keys,idx} | Flow | Entry,3)) |
| {keys,idx} | Flow | Entry,6)) |
| {keys,idx} | Flow | Entry,13)) |
| {keys,idx} | Flow | Entry,20)) |
| {keys,idx} | Flow | Entry,22)) |
| {retVal} | Flow | 1,Exit)) |
| {retVal} | Flow | 4,Exit)) |
| {key} | Flow | 6,7)) |
| {key} | Flow | 6,18)) |
| {cache} | Flow | 8,10)) |
| {result} | Flow | 9,15)) |
| {result} | Flow | 9,16)) |
| {it} | Flow | 10,11)) |
| {it} | Flow | 10,12)) |
| {entry} | Flow | 12,13)) |
| {entry} | Flow | 12,15)) |
| {mapped} | Flow | 13,14)) |
| {mapped} | Flow | 13,15)) |
| {entry} | Anti | 13,12)) |
| {mapped} | Anti | 14,13)) |
| {mapped} | Anti | 15,13)) |
| {retVal} | Flow | 16,Exit)) |
| {child} | Flow | 18,19)) |
| {child} | Flow | 18,22)) |
| {retVal} | Flow | 20,Exit)) |
| {retVal} | Flow | 22,Exit)) |

לאחר שברשותנו גרף PDG, נוכל לבחון את קודקודי הSDG הרי הם CONTROL LEAFS בגרף הPDG. נקבל את קבוצת הקודקודים:  
V = {1 2 4 5 6 8 9 10 12 13 15 16 17 18 20 21 22 23}

|  |  |
| --- | --- |
| Edge | Edge |
| (9,15) | (6,8) |
| (9,16) | (6,9) |
| (10,12) | (6,10) |
| (10,13) | (6,12) |
| (10,15) | (6,13) |
| (12,13) | (6,15) |
| (12,15) | (6,16) |
| (13,12) | (6,17) |
| (13,15) | (6,18) |
| (15,13) | (6,20) |
| (18,20) | (6,21) |
| (18,21) | (6,22) |
| (18,22) | (6,23) |
| (18,23) | (8,10) |



לאחר שברשותנו נמצא גרף הSDG, נוכל לבצע את חישוב הקבוצות: Before ו-After בהפעלת האלגוריתם עבור קבוצת קודקודים M:

.

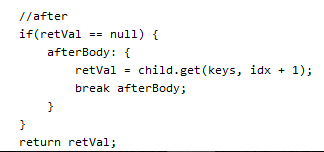
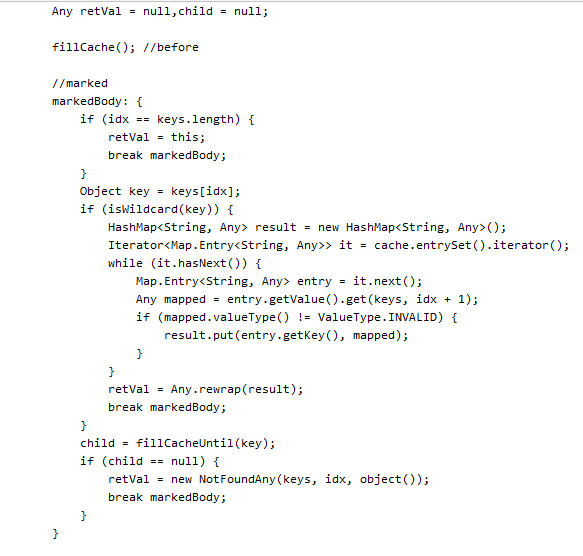
נבצע את חישוב הקבוצה reaching-M בעזרת אלגוריתם slides-first-search לאחור, ונראה כי נקבל את הקבוצה .  
לאחר מכן נבצע את חישוב הקבוצה M-Reachable בעזרת אלגוריתם slides-first-search קדימה, ונראה כי נקבל את הקבוצה:

תחילה נבצע חישוב עבור הקבוצה Marked:

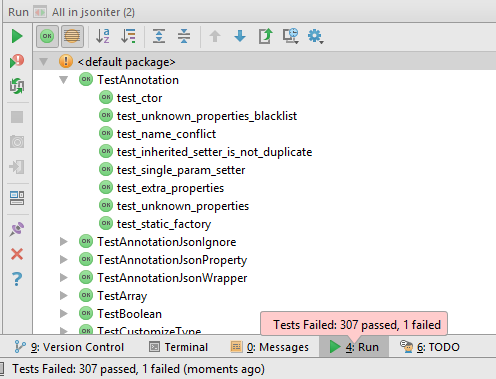
נחשב את קודקודי הBefore בעזרת האלגוריתם:

נחשב את קודקודי הAfter בעזרת האלגוריתם:

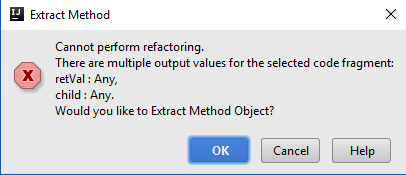
נתבונן בקטע קוד המתקבל, רגע לפני ביצוע Extract Method לשלושת החלקים Before, Marked ו-שAfter:



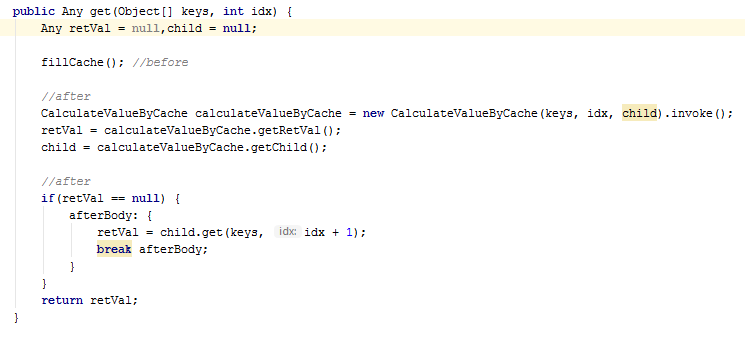
נשים לב שכעת, מכיוון שהקוד מחולק ל3 חלקים נפרדים, עלינו להוסיף תנאי בחלק הAfter שבודק האם בוצעה השמה לערך של retVal, מהסיבה כי בקוד המקורי, לאחר שבוצעה השמה לretVal הערך מוחזר מידית - ולכן רק במידה ולא בוצעה לו השמה – נרצה לעדכן אותו בהתאם. נראה שללא התנאי הנ"ל אחת מן הבדיקות נכשלת, כלומר התנאי הינו הכרחי לשימור נכונות התכנית:



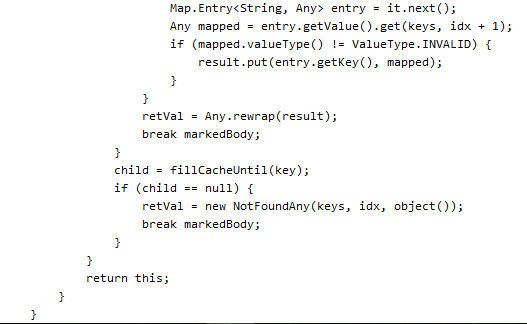
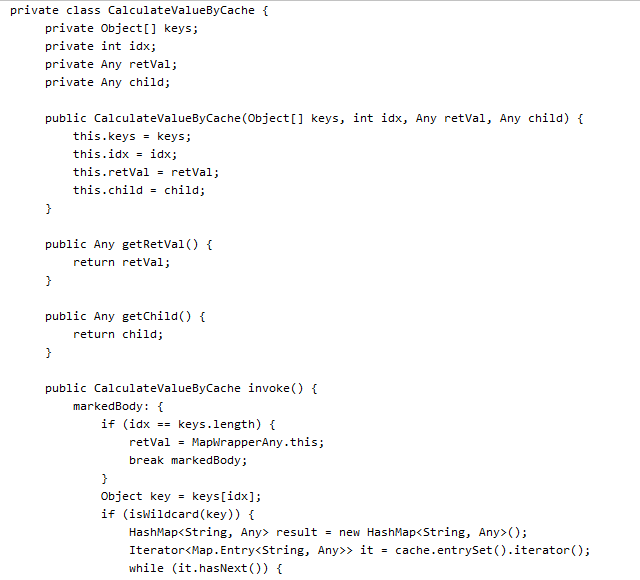
כעת, נרצה לבצע Extract Method לחלק הMarked, אולם אנו מקבלים מIntelliJ את ההודעה הבאה:



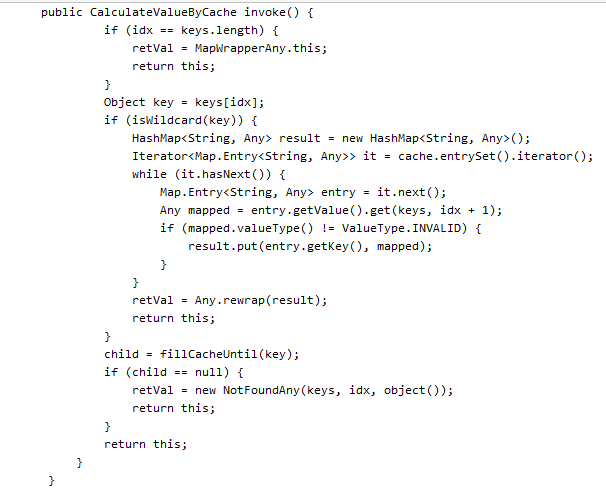
מדוע? נשים לב שבקטע קוד זה מבצעים השמה לשני משתנים שונים, לכן כדי ששניהם יוחזרו עם הערך המעודכן - מוצע לנו לאחסן אותם כאובייקט חדש. נלחץ OK, ונתבונן בתוצאה הבאה:



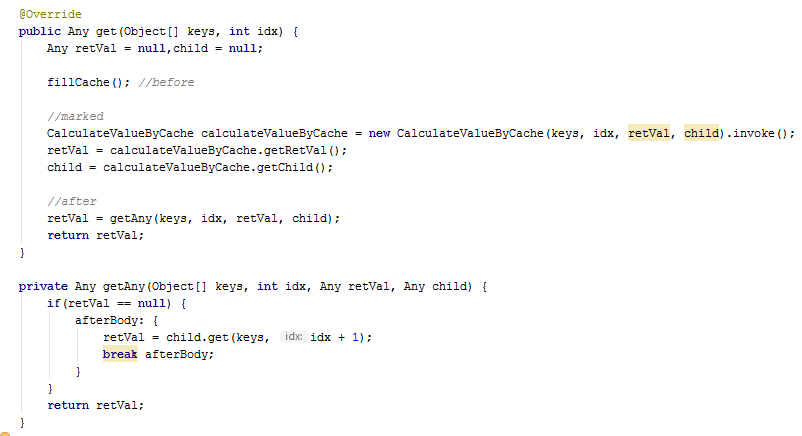
כמו כן נביט במחלקה שנוצרה עבור הrefactoring הנ"ל:



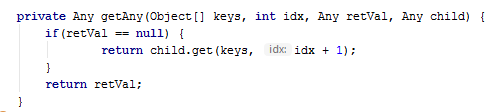
כעת נרצה להיפטר מכלל הlableים והbreakים אשר אין בהם צורך עוד – זאת בכדי לעבור לפורמט המקורי והנקי:



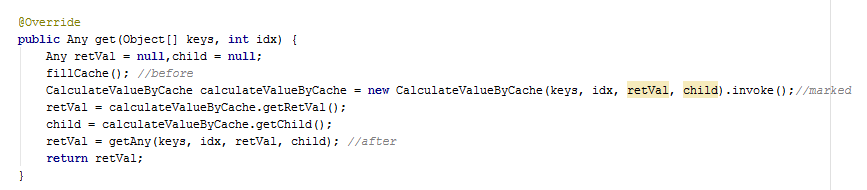
נבצע Extract Method לAfter:



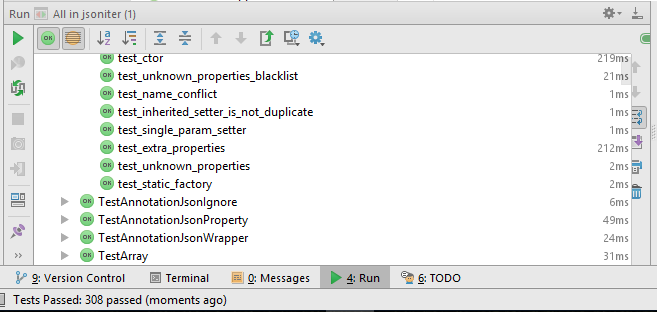
נוריד את הlabel והbreak כמקודם:



והתוצאה הסופית כדלהלן:



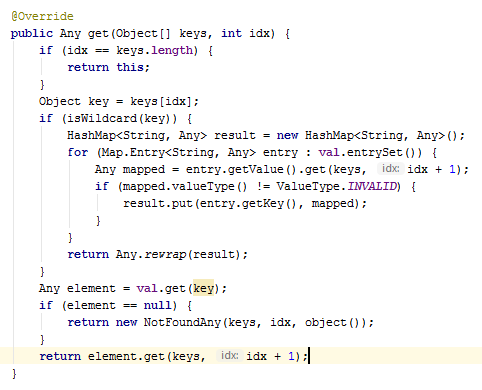
נריץ את כלל הטסטים כדי לוודא שהם עוברים:



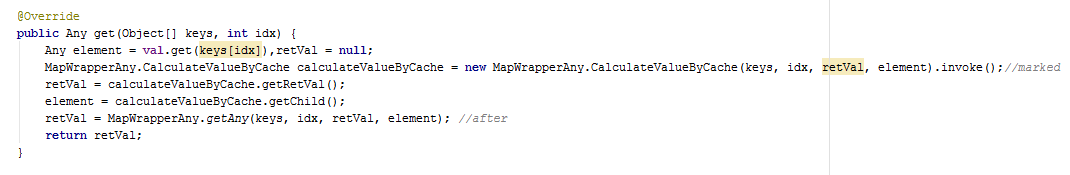
כנדרש.

כעת, ניזכר מלכתחילה מדוע בחרנו בקודקודים הבאים כMarked.  
במחלקה ObjectAny אותה ראינו בתחילת הדוגמה, קיים שכפול מסוג 3 אותו ראינו בתחילת דוגמה זו. נוכל כעת להשתמש בפונקציות להם עשינו Extract Method על מנת לחסוך בקוד (נשנה את הרשאות המחלקה CalculateValueByCache לציבורי, כמו כן את הפונקציה getAny() לציבורית וסטטית).

כעת הקוד המקורי הבא במחלקה ObjectAny:

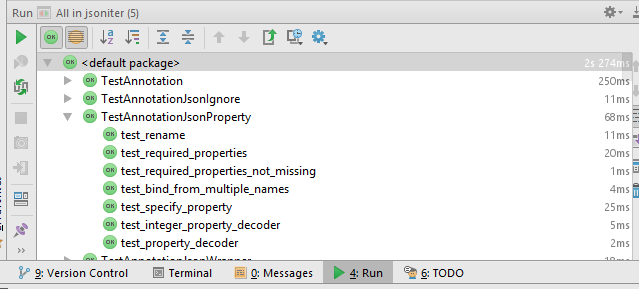


ניתן לrefactoring ומניעת קוד משוכפל, ע"י שימוש נכון בפונקציות אשר יצרנו בעזרת האלגוריתם



(נשים לב לשינוי בקוד הנ"ל - בו בחרנו להביר את element במקום child ונראה כי שאר הקוד אכן זהה לחלוטין)

כעת ניתן לראות בבירור כי אכן חסכנו בכתיבת קוד גם במחלקה אחרת בעזרת הrefactoring הנ"ל.  
כתוצאה מהשימוש באלגוריתם והיישום - הקוד קל יותר לתחזוקה וגמיש יותר – שכן ניתן לבצע שינויים ספציפיים במקום בודד מבלי לעבור על כל מחלקה אשר עושה שימוש בקוד. כמו כן הקוד עצמו כעת מובן יותר ופשוט לקריאה.



לא נשכח לוודא נכונות ע"י הרצת כלל הטסטים:

ובזאת נסיים את דוגמה זו.