מימוש בונוסים

בתרגיל מימשנו את שני הבונוסים שהוצגו בתרגיל.

- בדיקה כי אנו לא משתמשים במשתנה לוקאלי כאשר ייתכן ולא אותחל.
- בדיקה כי מוחזר ערך בכל control path, עבור כל מתודה שערך ההחזר שלה אינו void.

מקרי קצה

1. טיפול ב-Scopes: הסקופ הסטטי של מחלקה מופרד מן הסקופ הוירטואלי. לכן, בפרט מתודה סטטית ומתודה וירטואלית עם אותו השם בדיוק יכולות להיות מוגדרות באותה המחלקה (או ב- superclasses). את עניין הקריאה למתודה ללא prefix פתרנו כך, בתיאום עם המתרגל:

- אם הקריאה נעשית מתוך מתודה וירטואלית: תחילה נחפש מתודה סטטית עם שם זה (כלומר בכל הסקופ הסטטי, במחלקה הנוכחית וב-superclasses). אם קיימת כזו, הרי שזו המתודה אותה רצינו להפעיל. אם לא קיימת כזו, נחפש מתודה עם שם זה בסקופ הוירטואלי (כלומר בכל הסקופ הוירטואלי, במחלקה הנוכחית וב-superclasses. אם לא קיימת כזו, נזרוק שגיאה סמנטית).

- אם הקריאה נעשית מתוך מתודה סטטית, הרי שהמתודה אותה רוצים להפעיל יכולה להיות רק סטטית ולכן נחפש אותה רק בסקופ הסטטי.

2. דריסה חוקית : מאחר ולא מופיעה הגדרה של דריסה חוקית ב-SPEC. הנחנו כי יש להניח כי מדובר בדריסה זהה לזו שקיימת ב-JAVA. דהיינו, מספר הארגומנטים של המתודה הדורסת והטיפוסים שלהם חייבים להיות זהים בדיוק לאלה של המתודה הנדרסת. טיפוס ההחזר של המתודה הדורסת צריך להיות <u>תת טיפוס</u> של טיפוס ההחזר של המתודה הנדרסת.

כמו כן, בהנחיית המתרגל, אפשרנו דריסה של מתודות סטטיות, בדיוק לפי אותו פורמט.

3. בדיקת מתודת main: הנחנו שהכוונה בSPEC היא שבכל התוכנית יכולה להיות מוגדרת רק מתודה אחת עם השם main, ובנוסף לכך, הטיפוס שלה חייב להיות כפי שמוגדר ב-SPEC.

4. הדפסת ה-AST: בהתבסס על קבצי הקלט פלט שסופקו לנו, אנו מניחים שאין להדפיס את הספרייה כחלק מה-AST (גם כאשר מוסיפים אותה עם הדגל L–) כאשר משתמשים בדגל print-ast–

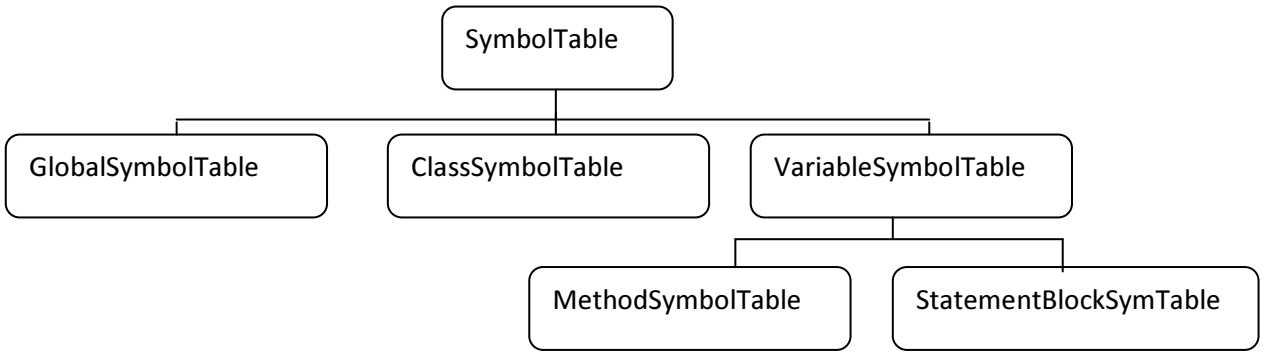
5. סדר הדגלים : מאחר והדבר אינו מוגדר בתרגיל, הנחנו שכאשר משתמשים בשני הדגלים print-ast, -dump-symtab– ביחד אזי ה-symbol tables וטבלת הטיפוסים יודפסו קודם ורק לאחר מכן יודפס ה-AST.

מבנה הקוד ומחלקות עיקריות

1. מחלקות עיקריות

1א. חבילה SymTables – כוללת את כל המחלקות שמייצגות ומטפלות בבניית symbol tables. כמו כן כוללת תת חבילה, Symbols, שמטפלת בייצוג ה-symbols (כניסות בתוך ה-symbol tables).

החבילה כוללת את ההיררכיה הבאה:



SymbolTable- מתארת symbolTable אבסטרקטי. מכילה מימוש משותף עבור כל הטבלאות: שם, טבלת הורה, טבלאות ילדים ועוד.

GlobalSymbolTable- מתארת את ה- SymbolTable הגלובלי שמכיל את ה-Symbols של כל המחלקות בתוכנית. כל הכניסות בטבלה זו הן מסוג ClassSymbol.

ClassSymbolTable- מתארת את ה-SymbolTable של מחלקה, מכילה סימנים של מתודות וירטואליות, מתודות סטטיות ושדות. מספקת הפרדה בין הסקופ הוירטואלי לסטטי. מספקת מתודות נוחות למציאת מתודות או שדות, מקומית או רקורסיבית (על ידי המשך החיפוש בהורים) לפי הסקופ - סטטי או וירטואלי, ולפי סוג הסימן- מתודה או שדה.

VariableSymbolTable- מימוש משותף אבסטרקטי של טבלה שלא מכילה סימנים של מתודות ומחלקות, אלא רק סימנים של משתנים. כוללת אוסף של סימנים שהם משתנים מקומיים [משותף לטבלה של מתודה וכן לטבלה של בלוק] וכן מתודות נוחות (לעיתים אבסטרקטיות) לחיפוש מקומי או רקורסיבי של משתנים ומתודות. בין היתר מכילה גם מתודה למציאת טיפוס המחלקה העוטפת (this) וטיפוס ההחזר של המתודה העוטפת.

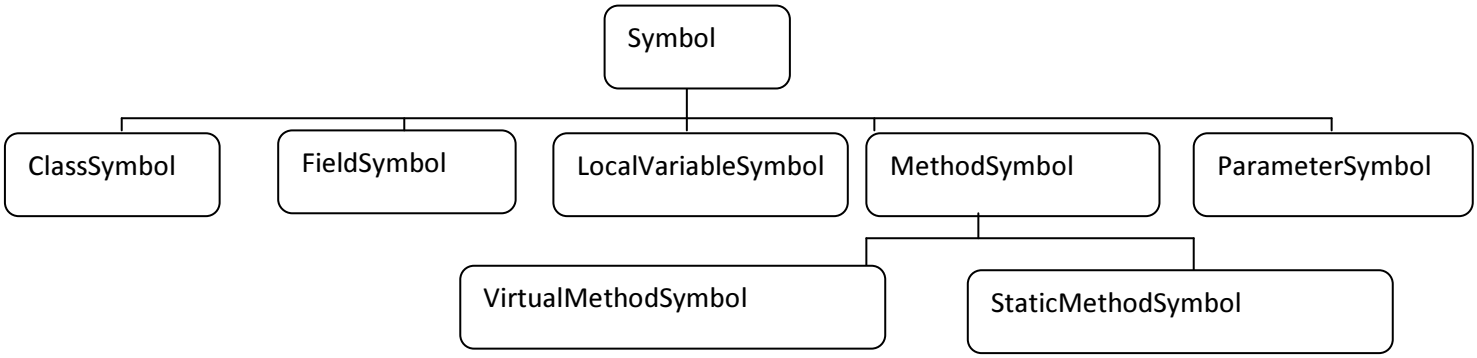
MethodSymbolTable- כוללת בנוסף לאוסף של משתנים לוקאליים גם אוסף פרמטרים. הסקופ (הטבלה) יכולה להיות סטטית ויכולה להיות וירטואלית (מכילה שדה isStatic). מממשת מתודות נוחות למציאת מתודות ומשתנים.

StatementBlockSymTable- מכילה את אוסף הסימנים שהם משתנים מקומיים (מקבלת בירושה מ-VariableSymbolTable) ומממשת מתודות הקשורות לחיפוש מתודות ומשתנים.

מחלקות נוספות שנמצאות בחבילה זו:

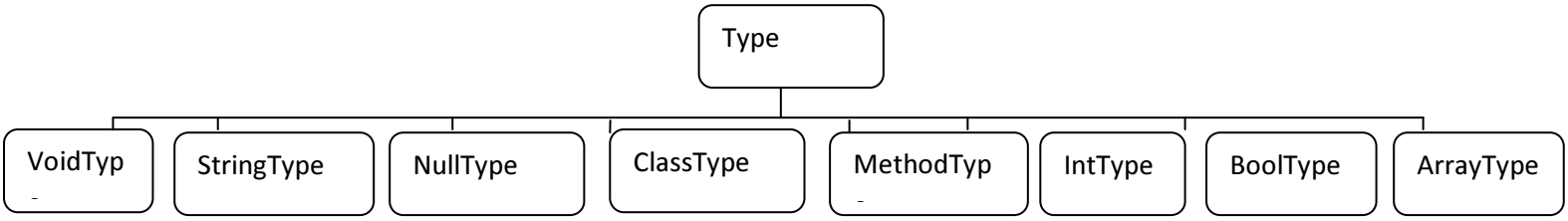
- *SymbolTableBuilder* – המבקר שעובר על ה-AST ובונה את ההיררכיה של הטבלאות.

1ב. חבילה Symbols (תת חבילה של SymTables) – כוללת ייצוג של Symbols לפי סוגיהם השונים (סימנים שהם כניסות ב-SymbolTable). החבילה כוללת את ההיררכיה הבאה:



כל מחלקה מתארת את הסימן המתאים לה, לרבות מכילה מימוש ייצוג מחרוזתי (toString) מתאים. לכל סימבול שומרים את המזהה שלו (שם) וטיפוס.

1ג. חבילת Types – מספקת מימוש של טבלת הטיפוסים (TypeTable), היררכית הטיפוסים וכן מבקרים שמבצעים את בדיקות הטיפוסים. ההיררכיה הבאה ממומשת בחבילה זו:



כל מחלקה מתארת את הטיפוס המתאים.

מחלקות חשובות נוספות:

- *TypeTable* – מממשת את טבלת הטיפוסים. מכילה מיפויים לכל סוג טיפוס, מתחזקת מונה טיפוסים, מספקת מתודות נוחות להוספת ומציאת טיפוסים.
- *SymbolSetter* – מבקר שרץ על ה-AST ומוסיף את כל הטיפוסים החדשים שהוא מוצא, לפי סדר הביקור, לטבלה. כמו כן, הוא מתאחל את הטיפוסים של ה-symbols באוסף הטבלאות symbol tables.

1ד. חבילת SemanticChecks- מכילה אוסף של מבקרים שממשמים בדיקות סמנטיות.

- *ChecksMainCorrectness* – מבקר שבודק כי קיימת מתודה יחידה שקוראים לה main, וכן כי היא בעלת חתימה נכונה.
- *InheritanceCheck* – מבקר שמבצע שהורשה מתבצעת בצורה נכונה וכי אין הסתרה של משתנים על ידי מתודות או להפך (וכן דריסה לא נכונה)

- *InitBeoforeUse* – מבקר שמממש את אחת הבדיקות של הבונוס. בודק כי נעשה שימוש במשתנים לוקאליים רק כאשר בוודאות ניתן לקבוע כי המשתנה מאותחל בנקודת השימוש.
- *methodReturnPaths* – מבקר שמממש את הבדיקה השנייה הבונוס: בכל מתודה שמחזירה ערך שהוא אינו VOID, כל מסלול control אפשרי מחזיר ערך.
- *StructuralChecks* – מימוש של הבדיקות כי continue, break מופעלים רק בתוך לולאות וכן כי this מופיעה רק בתוך מתודות וירטואליות.
- *SemanticError* – זריקה סמנטית שנזקרת על ידי המבקרים שמבצעים בדיקות סמנטיות. השגיאה נתפסת ב-IC.Compiler שמפעיל בדיקות אלה.

2. מימוש הבדיקות הסמנטיות וסדר הפעלתן

הבדיקות מופעלות על ידי המחלקה הראשית, IC.Compiler בסדר הבא:

- בשלב הראשון רץ המבקר שממומש ב-**ChecksMainCorrectness** על ה-AST, הוא בודק כי קיימת מתודה אחת ויחידה עם השם main, וכי החתימה שלה תואמת לדרישות התרגיל. כדי לעמוד בדרישות ההדפסות, המבקר מוסיף לטבלת הטיפוסים את הטיפוסים []string וכן טיפוס המתודה {string[]->void} אם הבדיקה מצליחה.
- בשלב השני רץ **SymbolTableBuilder** שבונה את ההיררכיה של ה-SymbolTables על סמך ה-AST. בזמן הבנייה, הוא מממש את הבדיקות הבאות: **1.** שום מזהה לא הוגדר פעמיים באותו סקופ **2**. כל משתנה לוקאלי הוגדר לפני השימוש בו. **3**. ההיררכיה של המחלקות היא עץ (אין forward extends או ירושה עצמית). נשים לב כי בשלב זה הסימנים בתוך הטבלאות הם חסרי טיפוס.
- בשלב השלישי רץ **SymbolSetter** שמוצא את כל הטיפוסים הייחודים בAST, לפי סדר הביקור, ומוסיף אותם לטבלת הטיפוסים. כמו כן, הוא מאתחל את הטיפוסים של הסימנים בטבלאות הסימנים שנבנו בשלב הקודם. כמו כן, ממומשת הבדיקה כי UserTypes מתייחסים למחלקות שאכן קיימות בתוכנית.
- שלב רביעי רץ **InheritanceCheck** שבודק כי הסקופים של מחלקות אינו מתנגש עם הסקופים של המחלקות מהן הן יורשות (הגדרות כפולות והסתרות). כמו כן מתבצעת בדיקה כי דריסה של פונקציות מתבצעת בצורה נכונה
- שלב חמישי רץ **StructuralChecks** שבודק כי continue, break מופיעים רק בתוך לולאות וכי this מופיעה רק בתוך מתודות וירטואליות.
- שלב שישי רץ **TypeEvaluator** שבודק כי כל ה-type rules המוגדרים ב-SPEC מתקיימים. כמו כן, המבקר מאתחל לכל ASTNode את הטיפוס שלו (פרט ל-statements).
- בשלב שביעי רץ **InitBeforeUse** – שמבצע את הבדיקה הרלבנטית של הבונוס, ראה 1ד.
- שלב שמיני רץ **methodReturnPaths** – שמבצע את הבדיקה הרלבנטית של הבונוס, ראה 1ד.

3. Testing Strategy – מימוש קבצי טסט

צירפנו את קבצי הטסט עמם בדקנו את נכונות הבדיקות (ואת קבצי הפלט המתאימים)

קבצים InitTest1 – InitTest17 – בודקים את הנכונות של InitBeforeUse.

קבצים main1 – main7 – בודקים את הנכונות של ChecksMainCorrectness.

קבצים return1 - return12 – בודקים את הנכונות של methodReturnPaths

קבצים sameScopeDef1 - sameScopeDef7 בודקים את הנכונות של SymbolTableBuilder.

קבצים structural1 – structural4 – בודקים את הנכונות של StructuralChecks

קבצים testInheritance1 – testInheritance14 – בודקים את הנכונות של InheritanceChecks

קבצים types1 – types17 – בודקים את הנכונות של TypeEvaluator וכן SymbolSetter

קבצים typeTable1, typeTable2 – בודקים את הנכונות מימוש של טבלת הטיפוסים וסדר ההכנסה.

הערה: הקבצים הם עם סיומת txt.