עץ ה Classes:

Class Hierarchy

- java.lang.Object
  - IC.AST.**ASTNode**

    node אבסטרקטי ממנו יורשים כל ה- nodes
    - IC.AST.**Expression**

      node אבסטקרטי לתיאור ביטויים
      - IC.AST.**BinaryOp**

        node אבסטקרטי לתיאור ביטויים מסוג פעולות בינאריות
        - IC.AST.**LogicalBinaryOp**

          node לתיאור ביטויים מסוג פעולות בינאריות לוגיות
        - IC.AST.**MathBinaryOp**

          node לתיאור ביטויים מסוג פעולות בינאריות מתמטיות
      - IC.AST.**Call**

        node אבסטקרטי לתיאור ביטויים מסוג קריאה למתודות
        - IC.AST.**StaticCall**

          node לתיאור קריאה למתודות סטטיות
        - IC.AST.**VirtualCall**

          node לתיאור קריאה למתודות וירטואליות (דינאמיות)
      - IC.AST.**ExpressionBlock**

        node לתיאור בלוק של ביטויים (בתור סוגריים)
      - IC.AST.**Literal**

        node לתיאור ביטויים מסוג ליטרלים
      - IC.AST.**Location**

        node אבסטקרטי לתיאור ביטויים מסוג מצביעים למשתנים
        - IC.AST.**ArrayLocation**

          node לתיאור ביטויים מסוג מצביע למערך
        - IC.AST.**VariableLocation**

          node לתיאור ביטויים מסוג מצביע למשתנה
      - IC.AST.**New**

        node אבסטקרטי לתיאור ביטויים מסוג יצירת אובייקטים
        - IC.AST.**NewArray**

          node לתיאור ביטויים מסוג יצירת מערך
        - IC.AST.**NewClass**

          node לתיאור ביטויים מסוג יצירת מחלקה
      - IC.AST.**This**

        node לתיאור הביטוי 'this'
      - IC.AST.**UnaryOp**

        node אבסטקרטי לתיאור ביטויים מסוג פעולות אונאריות
        - IC.AST.**LogicalUnaryOp**

          node לתיאור ביטויים מסוג פעולות אונאריות לוגיות

- IC.AST.**MathUnaryOp**

node לתיאור ביטויים מסוג פעולות אונאריות מתמטיות

- IC.AST.**Field**

node לתיאור שדה

- IC.AST.**Formal**

node לתיאור פרמטר של מתודה

- IC.AST.**ICClass**

node לתיאור הכרזה על מחלקה

- IC.AST.**Method**

node אבסטרקטי לתיאור מחלקה

  - IC.AST.**LibraryMethod**

  node לתיאור מחלקת ספרייה

  - IC.AST.**StaticMethod**

  node לתיאור מחלקה סטאטית

  - IC.AST.**VirtualMethod**

  node לתיאור מחלקה וירטואלית (דינאמית)

- IC.AST.**Program**

node שורש עבור תכנית IC

- IC.AST.**Statement**

node אבסטרקטי לתיאור פקודה

  - IC.AST.**Assignment**

  node לתיאור פקודת השמה

  - IC.AST.**Break**

  node לתיאור פקודת break

  - IC.AST.**CallStatement**

  node לתיאור פקודת קריאה למתודה

  - IC.AST.**Continue**

  node לתיאור פקודת continue

  - IC.AST.**If**

  node לתיאור פקודת תנאי if

  - IC.AST.**LocalVariable**

  node לתיאור פקודת הצהרה על משתנה מקומי

  - IC.AST.**Return**

  node לתיאור פקודת return

  - IC.AST.**StatementsBlock**

  node לתיאור בלוק של פקודות

  - IC.AST.**While**

  node לתיאור פקודת תנאי while

- IC.AST.**Type**

node אבסטרקטי לתיאור טיפוסי נתונים

  - IC.AST.**PrimitiveType**

  node לתיאור טיפוסי נתונים פרימיטיבים

- IC.**Compiler**

  המחלקה בעזרתה מריצים את התוכנית
- IC.Parser.**LibParser.CUP$LibParser$actions**
- java_cup.runtime.lr_parser
  - IC.Parser.**LibParser**
  - IC.Parser.**Parser**
- IC.SemanticChecks.**MainClassScopeChecker** (implements IC.SemanticChecks.IScopeCheck) בודק האם פונקצית המיין נקסע לפי כל הכללים
- IC.Parser.**Parser.CUP$Parser$actions**
- IC.AST.**PrettyPrinter** (implements IC.AST.Visitor) מבצע הדפסה עבור NODES של ה AST
- IC.Parser.**Scanner** (implements java_cup.runtime.Scanner)
- IC.SymbolTables.**Scope** -טיפוסי ה scopes של התוכנית
  - IC.SymbolTables.**BlockScope**

    נבנה כאשר עבור: ה

    nodes: While, If, StatementBlock - עבור הבנים
  - IC.SymbolTables.**ClassScope**

    נבנה עבור ה- ICCLASS
  - IC.SymbolTables.**GlobalScope**

    נבנה עבור ה- PROGRAM
  - IC.SymbolTables.**MethodScope**

    נבנה עבור ה- METHOD
  - IC.SymbolTables.**PrimitiveScope**
    - IC.SymbolTables.**FieldScope**

      נבנה עבור ה- FIELDS של ה- ICCLASS
    - IC.SymbolTables.**FormalScope**

      נבנה עבור הפרמטרים בהגדרת הפונקציה
    - IC.SymbolTables.**LocalScope**

      נבנה עבור Statement אחד שיכול להופיע ב- IF ו- WHILE
- IC.SemanticChecks.**ScopeCheck** (implements IC.SemanticChecks.ScopeVisitor)

  משתמש ב- MainCheck, ומבצע visit על ה- Scopes.
- IC.Parser.**sym**
- java_cup.runtime.Symbol
  - IC.Parser.**Token**
- IC.SymbolTables.**SymbolTable**

  טבלת ה- Symbols וה- Scopes
- IC.SymbolTables.**SymbolTablePrettyPrint** (implements IC.SemanticChecks.ScopeVisitor)

  הינו Visitor של היררכיית ה- Scopes, ומבצע הדפסה של ה- SymbolTable
- IC.SymbolTables.**SymbolTableVisitor** (implements IC.AST.Visitor)

  מבצע את הבנייה של ה- Symbol

- java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Error
        - IC.SemanticChecks.**SemanticError**

          נזרק בזמן ה- TypeCheck, כאשר קיים Semantic Error
- IC.SemanticChecks.**TypeCheck** (implements IC.AST.Visitor)

  מבצע visit על העץ, בודק ומבצע השמת SemanticTypes.
- IC.SymbolTables.**VirtualMap**

  מבנה של המתודות הוירטואליות.
- IC.SymbolTables.**VirtualMapVisitor** (implements IC.AST.Visitor)

  בונה את הVirtual Map.

<u>הסבר כללי:</u>

- קלט התוכנית הוא קובץ IC.
- ראשית פועלים ה- lexer וה- parser, שבונים את עץ ה-AST. כאשר שורש עץ ה-AST הוא node מסוג Program.
- לאחר מכן נבנה ה- SymbolTable:
  - מופעל ה- SymbolTableVisitor, אשר מקבל את שורש העץ (program כאמור) ומחזיר את המבנה SymbolTable (אוסף היררכי של ה- Scopes של התוכנית).
  - ה- SymbolTableVisitor מבצע פעולת visit עבור כל node בעץ הAST. כאשר בכל visit כזה הוא משייך ל- node ולבניו של אותו ה- node את ה- scope המתאים. בכל פעולה כזו ה- scope המתאים הינו ה- scope של האב, או scope חדש שיכנס לעץ ה- Scopes.
    למשל, כאשר מתבצע מעבר על בנים של node מסוג Assignment, ניקח את אותו ה- scope. עבור node מסוג While ייבנה scope חדש וה- scope של ה- While עצמו יוגדר כ- scope אב של הבן.
- לאחר מכן פועל ה- TypeCheck:
  - עבור כל node מתאים את הsemType בעץ ה- AST. ה- TypeCheck בודק האם כל ה- types מתאימים, האם ה- scopes נמצאים במקום המתאים, את הכללים של static ו- dynamic, ואת כללי ה- continue + break.
  - בדומה ל- SymbolType, מתבצע visit על כל node בעץ ונבדק מה ה- type שלו על ידי גישה ל- SymbolTable, ומעודכן ה- semantic type שלו.
  - כאשר ה- TypeCheck נמצא ב- node ורוצה לבדוק מה ה- type שלו,הוא מברר את ה- type ע"י ה- Literals או ע"י ה- ID. חיפוש ה- ID המתאים מתבצע ב- scope שלו או ב- scopes של האבות שלו ע"י המתודה: getSymbol של הSymbolTable.
  - בכל שלב ה- visit יבדוק האם זמו ה- Assignment, או הReturn, הtypes מתאימים. אחרת ייזרק ה- exception: SemanticError, עם ה- error message המתאים, שייתפס ויכתב ב- main של מחלקת ה- 'Compiler'.
- לאחר מכן יפעל ה- MainClassScopeChecker:
  - בדומה ל- TreeVisitor ישנו גם ScopeVisitor שעובד בצורה דומה. הוא מבצע visit בכל ה- scopes ומפעיל פעולה מסויימת.
  - במקרה הנ"ל הפעולה הינה MainClassScopeChecker. מתבצעת בדיקה האם ה- scope הינו מטיפוס method, ואם כן ימנה ויבדוק לפי כללי ה- SPEC את נכונתה וקיומה של הפונקציה main.