Extending LINVAST with Java

Dara Milojković
Marija Katić

Uvod

LINVAST je biblioteka koja pruža podršku kreiranja **jezički invarijantnog AST-a** za mnoge programske jezike. AST se kreira apstrahovahovanjem stabla parsiranja koji se dobija pomoću alata ANTLR iz gramatike jezika. Trenutno postoji podrška za programske jezike **C** i **Lua**.

Cilj ovog projekta je dodati podršku za programski jezik **Java**.

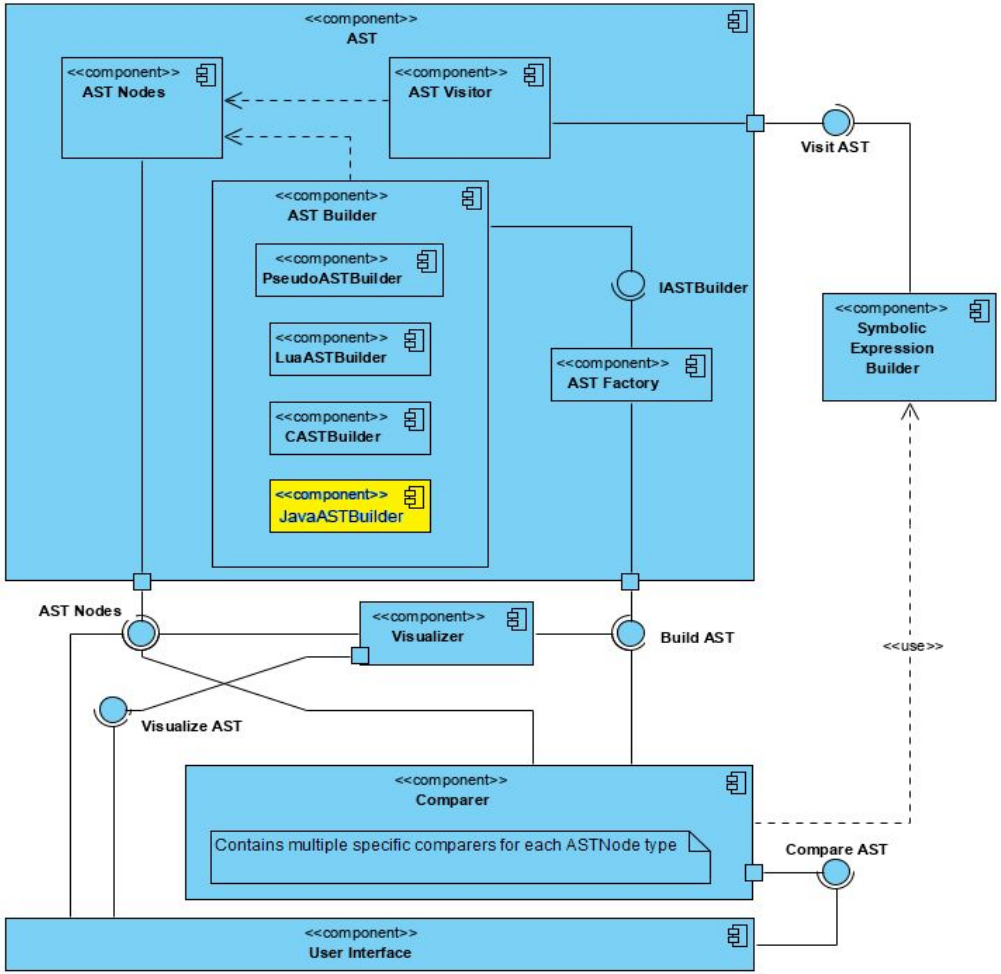
Posao na projektu je prirodno bio podeljen na obradu:

- Deklaracija
- Tipova
- Izraza
- Funkcija

Odnos našeg projekta i celog sistema

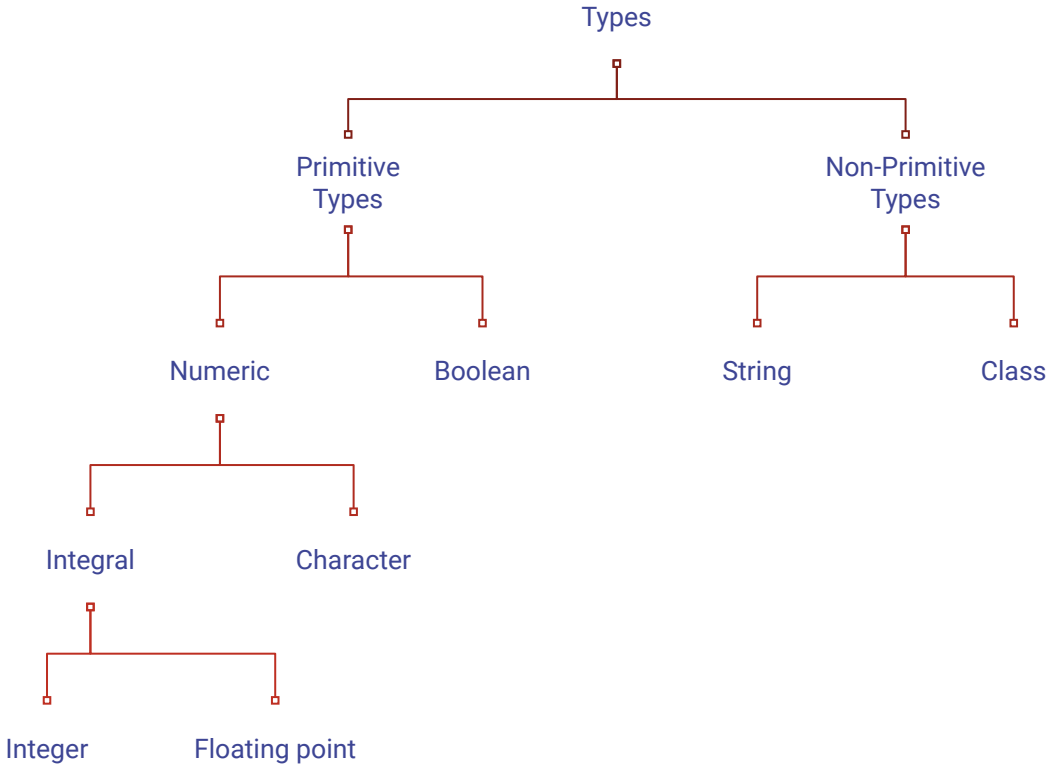
Zadatak je bio implementirati **JavaASTBuilder** - “Bilder” koji stablo parsiranja jezika Java prevodi u naš jezički invarijatni AST.

Kao i dosadašnji “Bilderi” (PseudoASTBuilder, LuaASTBuilder, CASTBuilder), JavaASTBuilder **implementira ASTBuilder**, i **nasleđuje ANTLR-ovu Base Visitor klasu**.



Java Types

```
int x;
Object o;
char c;
Array<Integer> a;
void function(float x);
```



ANTLR4 primer tipova

```
typeDeclaration
    : classOrInterfaceModifier*
      (classDeclaration | enumDeclaration | interfaceDeclaration | annotationTypeDeclaration)
    | ';'
    ;

classOrInterfaceModifier
    : annotation
    | PUBLIC
    | PROTECTED
    | PRIVATE
    | STATIC
    | ABSTRACT
    | FINAL    // FINAL for class only -- does not apply to interfaces
    | STRICTFP
    ;
```

```
classType
    : (classOrInterfaceType '.')? annotation* IDENTIFIER typeArguments?
    ;
```

A.B<C>.D varId<T>

```
public override ASTNode VisitClassType([NotNull] ClassTypeContext ctx)
{
    if (ctx.annotation().Any()) {
        throw new NotImplementedException("annotations");
    }

    int ctxStartLine = ctx.Start.Line;

    TypeNameListNode templlist = new TypeNameListNode(ctxStartLine), baselist = new TypeNameListNode(ctxStartLine);
    if (ctx.classOrInterfaceType() is { }) {
        TypeNameNode typeName = this.Visit(ctx.classOrInterfaceType()).As<TypeNameNode>();

        ctxStartLine = ctx.classOrInterfaceType().Start.Line;

        baselist = new TypeNameListNode(ctxStartLine, typeName);
    }

    if (ctx.typeArguments() is { }) {

        templlist = this.Visit(ctx.typeArguments()).As<TypeNameListNode>();
    }

    var identifier = new IdNode(ctxStartLine, ctx.IDENTIFIER().GetText());
    return new TypeDeclNode(ctxStartLine, identifier, templlist, baselist, new ArrayList<DeclStatNode>());
}
```

```
primitiveType
    : BOOLEAN
    | CHAR
    | BYTE
    | SHORT
    | INT
    | LONG
    | FLOAT
    | DOUBLE
    ;
```

```
public override ASTNode VisitPrimitiveType([NotNull] PrimitiveTypeContext ctx)
    => new TypeNameNode(ctx.Start.Line, ctx.children.First().GetText());
```

Types - Problemi

```
typeArguments
    : '<' typeArgument (',' typeArgument)* '>'
    ;

annotation
    : ('@' qualifiedName | altAnnotationQualifiedName) ('(' ( elementValuePairs | elementValue )? ')')?
    ;
```

Object<T_1, T_2, T_3, T_4, …, T_n> o;
<S extends Comparable<T>>
public class Employee implements java.io.Serializable

Declarations

```
compilationUnit
    : packageDeclaration? importDeclaration* typeDeclaration* EOF
    ;
```

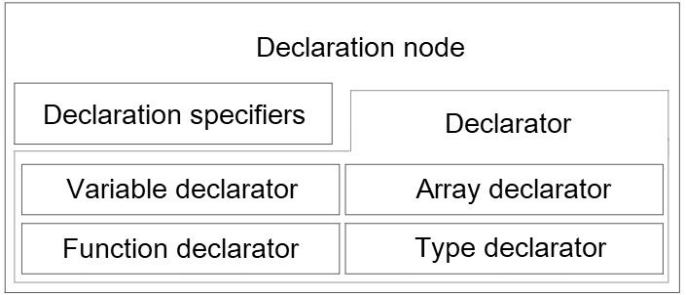
```
typeDeclaration
    : classOrInterfaceModifier*
      (classDeclaration | enumDeclaration | interfaceDeclaration |
annotationTypeDeclaration)
    | ';'
    ;
```

Declarations

- Package Declaration
- Import Declaration
- Class Declaration
- Enum Declaration
- Interface Declaration
- Annotation Type Declaration
- Class/Interface Member Declaration
  - Field Declaration
  - Method Declaration
  - ...
- (Local) Variable Declarations

Declarations - problemi (1)

“Nekompatibilnosti” između ANTLR-ovog stabla parsiranja i našeg jezički invarijantnog AST-a



Declaration specifiers

=

Modifiers + Type Name

```
1  extern static const int x = 3, arr[] = {1, 2, 3};
2
3  public static final int x = 3;
4  public static final int[] arr = new int[] {1, 2, 3};
5
6  public static readonly int x = 3;
7  public static readonly int[] arr = new[] {1, 2, 3};
```

Declaration specifiers — Declarator — Identificator — Initializer

Declarations - problemi (1)

```
classBodyDeclaration
: ';'
| STATIC ? block
| modifier* memberDeclaration
;
memberDeclaration
:
   ...
| fieldDeclaration
   ...
;
```

```
fieldDeclaration
: typeType variableDeclarators
';'
;
variableDeclarators
: variableDeclarator (','
variableDeclarator )*
;
variableDeclarator
: variableDeclaratorId ('='
variableInitializer )?
;
```

Declarations - problemi (2)

```
classDeclaration
    : CLASS IDENTIFIER typeParameters?
      (EXTENDS typeType)?
      (IMPLEMENTS typeList)?
      classBody
    ;
```

- Dok u našem AST-u postoji samo lista Base Types, u kojoj čuvamo i nasleđene klase i implementirane interfejse
- Odlučeno je da je u redu typeType i typeList spojiti u jednu listu BaseTypes u našem AST-u

Declarations - problemi (3)

Kako uklopiti u dosadašnje AST stablo neke novine koje donosi OOP?

Na primer:

- **Konstruktori** - Funkcije bez povratne vrednosti
- **Static blokovi unutar klasa** - Funkcije bez povratne vrednosti i bez naziva
- **Non-static blokovi unutar klasa** - Isto kao static blokove

Sporedni efekti rada na projektu

Vrlo detaljno izučena gramatika programskog jezika Java. :)

Zanimljivi primeri:

- Osim static blokova koji se mogu deklarisati unutar tela klase (koji se izvršavaju u vreme učitavanja klase), postoje i non-static blokovi (oni se izvršavaju u vreme kreiranja novog objekta, pre konstruktora). Deklarišu se ostavljanjem bloka unutar tela klase, bez ikakvih ključnih reči.
- Kada deklarišmo niz, zagrade [ ] možemo ostaviti i nakon imena promenljive, iako skoro uvek viđamo da stoje nakon naziva tipa.
- Moguće je da zagrade [ ] stoje iza definicije funkcije, i tada se vezuju za tip povratne vrednosti, i označavaju da funkcija vraća niz tog tipa.

Primeri - AST

Hvala na paznji

Zahvaljujemo Ivanu Ristoviću na podršci , pomoći i trudu