```java
package types;

import symtable.Type;

public class ArrayType implements Type {
  int count;
  Type subType;

  public ArrayType(int count, Type subType) {
    this.count = count;
    this.subType = subType;
  }

  @Override
  public String toString() {
    StringBuilder typeStr = new StringBuilder();
    if (count == 0) {
      return typeStr.append(subType).toString();
    }
    return typeStr.append("array(")
        .append(count)
        .append(",")
        .append(subType)
        .toString();
  }
}
```

```java
package types;

import org.antlr.v4.runtime.tree.ParseTreeProperty;

import java.util.HashMap;
import java.util.Map;

import symtable.BasicTypeSymbol;
import symtable.Type;
import symtable.VariableSymbol;

public class TypeCheckingListener extends ArrayBaseListener {
  private final Map<String, VariableSymbol> symbolTable = new HashMap<>();
  private final ParseTreeProperty<Type> arrayTypeProperty = new ParseTreeProperty<>();
  private final ParseTreeProperty<Type> basicTypeProperty = new ParseTreeProperty<>();

  /**
   * Pass the basic type from top to bottom
   */
  @Override
  public void enterArrDecl(ArrayParser.ArrDeclContext ctx) {
    String typeName = ctx.type().getText();
    Type basicType = new BasicTypeSymbol(typeName);
    basicTypeProperty.put(ctx, basicType);
  }

  @Override
  public void enterNonEmptyArrayType(ArrayParser.NonEmptyArrayTypeContext ctx) {
    basicTypeProperty.put(ctx, basicTypeProperty.get(ctx.parent));
  }

  @Override
  public void enterEmptyArrayType(ArrayParser.EmptyArrayTypeContext ctx) {
    basicTypeProperty.put(ctx, basicTypeProperty.get(ctx.parent));
  }

  /**
   * Below: construct the array type from bottom to top
   */

  @Override
  public void exitEmptyArrayType(ArrayParser.EmptyArrayTypeContext ctx) {
    arrayTypeProperty.put(ctx, new ArrayType(0, basicTypeProperty.get(ctx.parent)));
  }

  @Override
```

```
47   public void exitNonEmptyArrayType(ArrayParser.
   NonEmptyArrayTypeContext ctx) {
48     int dimension = Integer.parseInt(ctx.INT().getText());
49     Type subArrayType = arrayTypeProperty.get(ctx.arrayType());
50
51     Type arrayType = new ArrayType(dimension, subArrayType);
52     this.arrayTypeProperty.put(ctx, arrayType);
53   }
54
55   @Override
56   public void exitArrDecl(ArrayParser.ArrDeclContext ctx) {
57     Type arrayType = arrayTypeProperty.get(ctx.arrayType());
58     arrayTypeProperty.put(ctx, arrayType);
59
60     String arrayName = ctx.ID().getText();
61     symbolTable.put(arrayName, new VariableSymbol(arrayName, arrayType
   ));
62   }
63
64   @Override
65   public void exitArrDeclStat(ArrayParser.ArrDeclStatContext ctx) {
66     System.out.println("ArrayType : " + arrayTypeProperty.get(ctx.
   arrDecl()));
67   }
68
69   /**
70    * Below: type reference and inference
71    */
72   @Override
73   public void exitId(ArrayParser.IdContext ctx) {
74     arrayTypeProperty.put(ctx, symbolTable.get(ctx.ID().getText()).
   getType());
75   }
76
77   @Override
78   public void exitInt(ArrayParser.IntContext ctx) {
79     arrayTypeProperty.put(ctx, new BasicTypeSymbol("int"));
80   }
81
82   @Override
83   public void exitVarDecl(ArrayParser.VarDeclContext ctx) {
84     String varName = ctx.ID().getText();
85     String typeName = ctx.type().getText();
86     Type type = new BasicTypeSymbol(typeName);
87
88     symbolTable.put(varName, new VariableSymbol(varName, type));
89   }
90
91   // type inference
92   @Override
93   public void exitArrayIndex(ArrayParser.ArrayIndexContext ctx) {
94     arrayTypeProperty.put(ctx,
95         ((ArrayType) arrayTypeProperty.get(ctx.primary)).subType);
```

```
 96   }
 97
 98   /**
 99    * Below: assign
100    */
101   @Override
102   public void exitAssignStat(ArrayParser.AssignStatContext ctx) {
103     Type lhs = arrayTypeProperty.get(ctx.lhs);
104     Type rhs = arrayTypeProperty.get(ctx.rhs);
105     System.out.println(lhs + " : " + rhs);
106   }
107 }
```