Syntax Tree

Abstract Syntax Tree

Implementation via Abstract Classes in Java

Abstract Classes in Java: Definition

- Abstract classes may or may not contain abstract methods, i.e., methods without body
  - `public abstract void get();`

- If a class has at least one abstract method, then the class must be declared abstract.

- If a class is declared abstract, it cannot be instantiated.

- To use an abstract class A, another class B inherits from A, and B provides implementations to the abstract methods in it.

Example

Download and compile/run the example code for Employee.java and AbstractDemo.java. Do this in a terminal window.

```
cd [directory files are installed]
javac Employee.java
javac AbstractDemo.java
java AbstractDemo
```

What happens?

Examine code for Employee/AbstractDemo

- The Employee class has three fields, seven methods and one constructor.

- We cannot create an Employee object because the class is abstract.

- We use the Employee class through inheritance – subclasses that inherit the structure of the Employee class.

Fixing the Instantiation Problem

Next, we define a child class of Employee, Salary class, and compile a second version of the AbstractDemo2.java code

```
javac Employee.java Salary.java AbstractDemo2.java
java AbstractDemo2
```

Examine Code for Salary/AbstractDemo2

```
public Salary(String name, String address, int number, double salary)
{
    super(name, address, number); //use Employee constructor
    setSalary(salary);
}


public void mailCheck() {
    System.out.println("Within mailCheck of Salary class ");
    System.out.println("Mailing check to " + getName() + " with
salary " + salary);  //use Employee method getName()
}
```

Abstract methods

- If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as abstract.
- This is a case of using inheritance for specification.
- Example - Add this abstract method to Employee.java:

```
public abstract class Employee {
   private String name;
   private String address;
   private int number;
   public abstract double computePay();
   // Remainder of class definition
}
```

Abstract methods

Declaring a method as abstract has two consequences −

- The class containing it must be declared as abstract.
- Any class inheriting the current class must either override the abstract method or declare itself as abstract.

Modify the example

- Modify Salary.java as follows

```
/* File name : Salary.java */
public class Salary extends Employee {
   private double salary;   // Annual salary
   public double computePay() {
      System.out.println("Computing salary pay
for " + getName());
      return salary/52;
   }
   // Remainder of class definition
}
```

- Call computePay for one of the instantiated Salary objects

Why Abstract Classes?

- The next phase in creating a parser for PDef requires the construction of an Abstract Syntax Tree.
- This construction is best done using inheritance and abstract classes.

PDef-Lite Grammar Rules

Program	→ Block eofT
Block	→ lcbT StmtList rcbT
StmtList	→ Stmt { commaT Stmt }
Stmt	→ Declaration | Assignment | Block
Declaration	→ typeT identT
Assignment	→ identT assignT identT

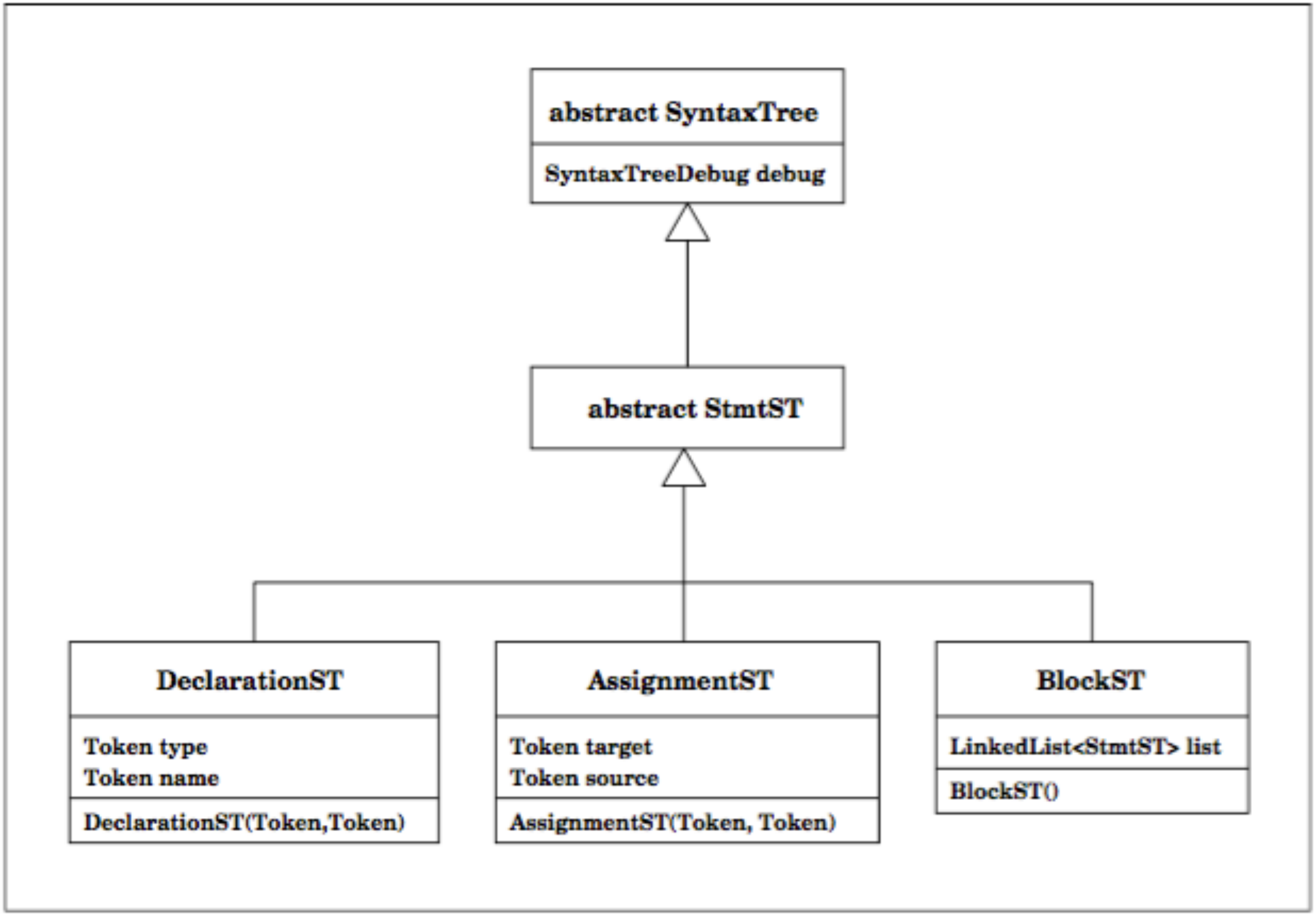


Figure 14.3: UML Class Diagram of the Syntax Tree for PDef-*lite*

AST

Classes in Code Distributed for Project 3

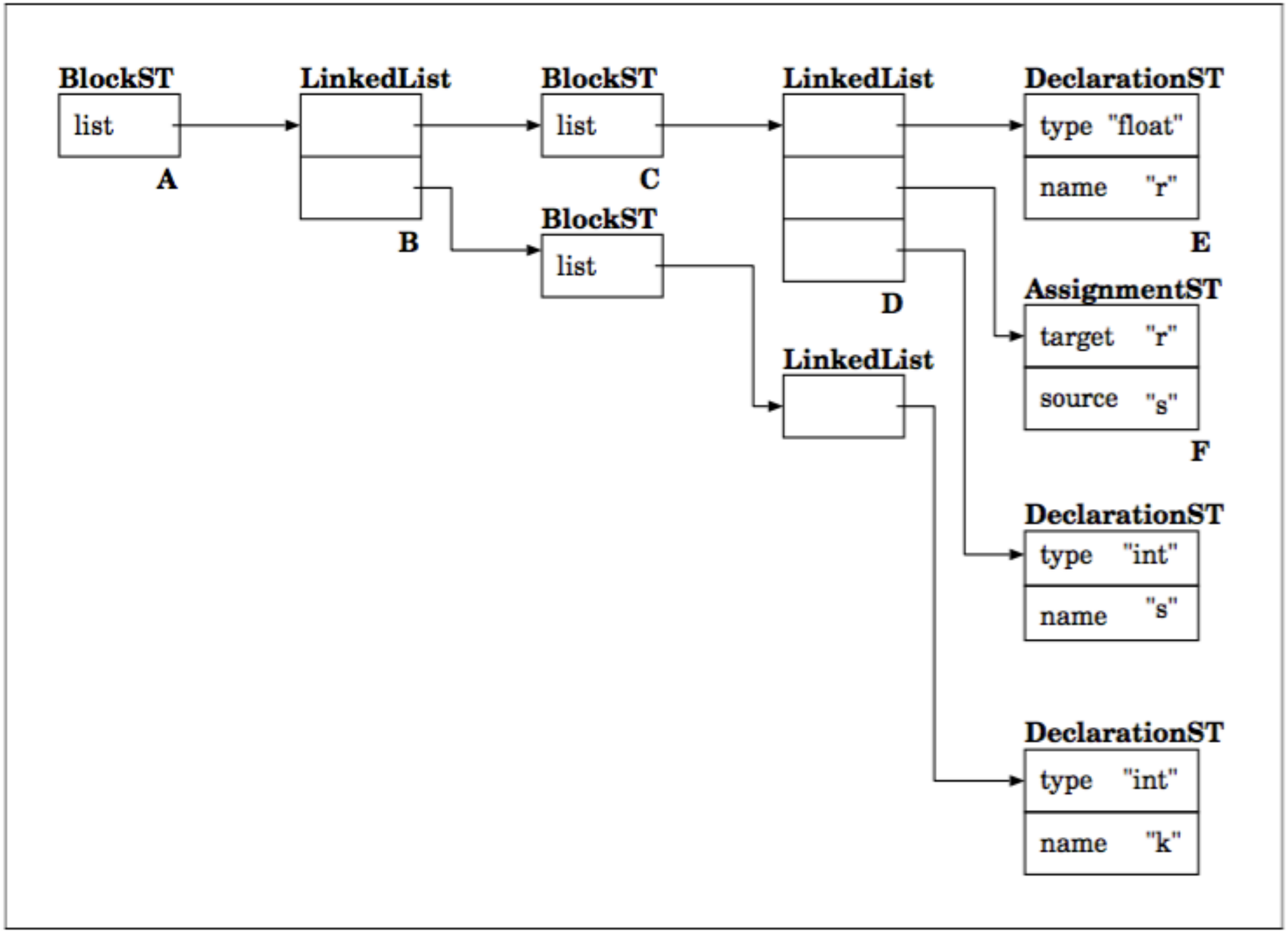
```
public abstract class SyntaxTree {
       protected SyntaxTreeDebug debug = new SyntaxTreeDebug();
}

public abstract class StmtST extends SyntaxTree { }

public class BlockST extends SyntaxTree {
 private LinkedList<StmtST> list;
 public BlockST(LinkedList<StmtST> l) { list=l; }
 public void traverseST() {
           // for (StmtST st : list)
           //     st.traverseST();
           System.out.println("BlockST");
} }
```

Example: PDef program and AST

{ { float r, r = s, int s }, { int k } } - { { X1, X2, X3 }, { Y1 } }



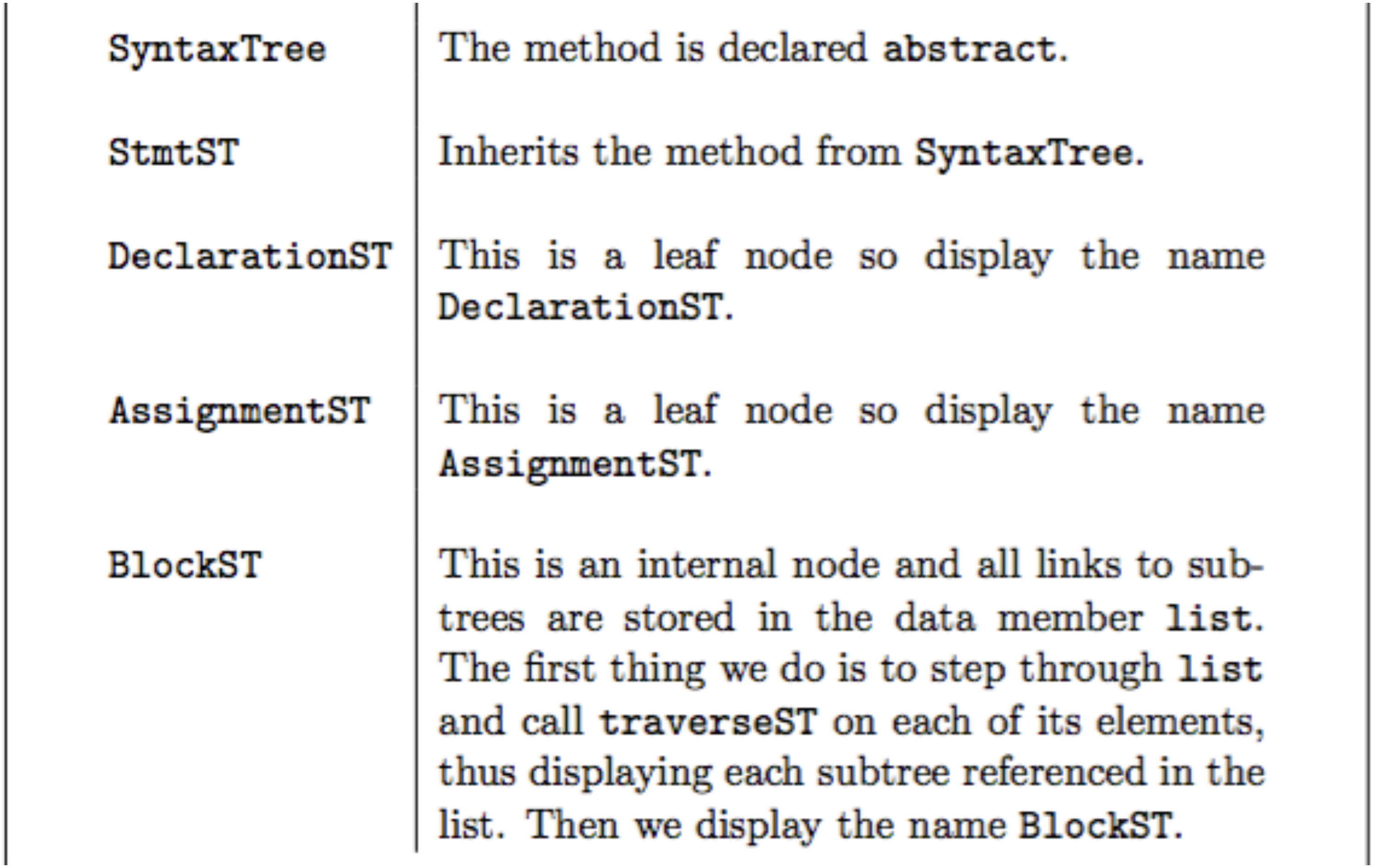
AST

Traversing the AST

```
// code for DeclarationST.traverseST
public void traverseST() {
   System.out.println("DeclarationST");
}

// code for BlockST.traverseST
public void traverseST() {
   for (StmtST st : list)
       st.traverseST();
   System.out.println("BlockST");
}
```

Traversing the AST - traverseST



`SyntaxTree`	The method is declared `abstract`.
`StmtST`	Inherits the method from `SyntaxTree`.
`DeclarationST`	This is a leaf node so display the name `DeclarationST`.
`AssignmentST`	This is a leaf node so display the name `AssignmentST`.
`BlockST`	This is an internal node and all links to subtrees are stored in the data member `list`. The first thing we do is to step through `list` and call `traverseST` on each of its elements, thus displaying each subtree referenced in the list. Then we display the name `BlockST`.

PDef-Lite parser in action

```
{ int a, float b, { a = b, { int x, a = x }, { b = a } }, a = b }
```

```
Program parsed!
Here's the Syntax Tree
DeclarationST
DeclarationST
AssignmentST
DeclarationST
AssignmentST
BlockST
AssignmentST
BlockST
BlockST
AssignmentST
BlockST
```