Paul Kummer
Dr. Hanku Lee
Compilers CSIS 455-01
09/16/2021

Lab 03: Exploring AST with Visitors

Purpose of Lab

The purpose of this lab is to show how an abstract syntax tree can be traversed down through its nodes. Also, it shows how information about the tree traversals can be stored as the traversal occurs. Another important concept that is covered is the order that the traversals occur.

Example Code

Elements

The elements in the code represent nodes of an AST. Some nodes contain other nodes, which can be accessed with the dot notations.

ElementRoot.java

```
1   public class ElementRoot implements ElementNode
2   {
3       ElementNode root;
4
5       public ElementRoot()
6       {
7           root = new ElementA();
8       }
9
10      @Override
11      public void accept(ElementVisitor visitor)
12      {
13          visitor.visit(this);
14
15          //root.accept(visitor);
16      }
17  }
```

ElementA.java

```
1   public class ElementA implements ElementNode
2   {
3       ElementNode b = new ElementB();
4       ElementNode c = new ElementC();
5
6       @Override
7       public void accept(ElementVisitor visitor)
8       {
9           visitor.visit(this);
10      }
11  }
```

ElementB.java

```java
public class ElementB implements ElementNode
{
    ElementNode d = new ElementD();
    ElementNode e = new ElementE();

    @Override
    public void accept(ElementVisitor visitor)
    {
        visitor.visit(this);
    }
}
```

ElementC.java

```java
public class ElementC implements ElementNode
{
    ElementNode f = new ElementF();

    @Override
    public void accept(ElementVisitor visitor)
    {
        visitor.visit(this);
    }
}
```

ElementD.java

```java
public class ElementD implements ElementNode
{

    @Override
    public void accept(ElementVisitor visitor)
    {
        visitor.visit(this);
    }
}
```

ElementE.java

```java
public class ElementE implements ElementNode
{

    @Override
    public void accept(ElementVisitor visitor)
    {
        visitor.visit(this);
    }
}
```

ElementF.java

```java
public class ElementF implements ElementNode
{

    @Override
    public void accept(ElementVisitor visitor)
    {
        visitor.visit(this);
    }
}
```

Visitors

The visitor's code is the generic code for any node, which is passed as a parameter to other nodes. Then the visitor code will execute different code depending on what the visitor object's signature is. The display visitor will output to the terminal how the nodes are being visited. The level of the tree is also maintained by incrementing or decrementing a level counter before and after returning from a node.

ElementDisplayVisitor.java

```java
public class ElementDisplayVisitor implements ElementVisitor
{
    int level = 0;
    @Override
    public void visit(ElementRoot r)
    {
        System.out.println("Element Root: " + level);

        level++;
        r.root.accept(this);
        level--;
    }

    @Override
    public void visit(ElementA a)
    {
        System.out.println("Element A: " + level);

        level++;
        a.b.accept(this);
        a.c.accept(this);
        level --;
    }

    @Override
    public void visit(ElementB b)
    {
        System.out.println("Element B: " + level);

        level++;
        b.d.accept(this);
        b.e.accept(this);
        level--;
    }

    @Override
    public void visit(ElementC c)
    {
        System.out.println("Element C: " + level);

        level++;
        c.f.accept(this);
        level--;
    }

    @Override
    public void visit(ElementD d)
    {
        System.out.println("Element D: " + level);
    }

    @Override
    public void visit(ElementE e)
    {
        System.out.println("Element E: " + level);
    }

    @Override
    public void visit(ElementF f)
    {
        System.out.println("Element F: " + level);
    }
}
```

ElementVisitor.java

```
1   public interface ElementVisitor
2   {
3       public void visit(ElementRoot r);
4       public void visit(ElementA a);
5       public void visit(ElementB b);
6       public void visit(ElementC c);
7       public void visit(ElementD d);
8       public void visit(ElementE e);
9       public void visit(ElementF f);
10  }
```

Executable

The executable begins the abstract tree traversal by creating a root node that will call visit and accept methods for each node. When this happens the display element code will execute, showing each node as it is visited and also show what level of the AST it is at.

ElementDemo.java

```
1   public class ElementDemo
2   {
3       public static void main(String[] args)
4       {
5           ElementNode root = new ElementRoot();
6           root.accept(new ElementDisplayVisitor());
7       }
8   }
```

```
cx3645kg@smaug:~/Documents/CSIS455_compilers/lab03$ javac *.java
cx3645kg@smaug:~/Documents/CSIS455_compilers/lab03$ java ElementDemo
Element Root: 0
Element A: 1
Element B: 2
Element D: 3
Element E: 3
Element C: 2
Element F: 3
cx3645kg@smaug:~/Documents/CSIS455_compilers/lab03$
```

Conclusion

Visitors are an effective way to traverse an abstract syntax tree. They allow flexibility of controlling when operations will be performed during tree traversals. This can be important if information is being gathered for performing prefix, infix, or postfix expressions. Additionally, this lab showed how information can be passed down to nodes by the parent node changing global values before it gets to the child.