COM S-342

Recitation 09/10/18 – 09/12/18

Today

- Short introduction to ANTLR
- Visitor Pattern

ANTLR

- ANTLR (ANother Tool for Language Recognition) is parser generator
- ANTLR generates a parser from the grammar
- It can be used to walk the parse trees, or
- Generate an abstract syntax tree

Grammar Example

- Define a simple data structure representing a Json tree
- Keys are strings and values can be strings, numbers (integers) or Json objects
- Use ANTLR to generate a Json tree
- Example: {"Set":{"I am":"here", "You are":"cow"}, "world":313, "bye":"home"}

Grammar Example

```java
public abstract class JsonTree {

    public abstract Object accept(Visitor visitor);

    public static class Str extends JsonTree {
        String str;
        public Str(String str) { this.str = str; }
        public String getStr() {
            return str;
        }
        @Override
        public Object accept(Visitor visitor) { return visitor.visit( e: this); }
    }

    public static class Num extends JsonTree {
        int num;
        public Num(int num) { this.num = num; }
        public int getNum() { return num; }
        @Override
        public Object accept(Visitor visitor) { return visitor.visit( e: this); }
    }

    public static class Dict extends JsonTree {
        List<Pair> pairs;
        public Dict(List<Pair> pairs) {
            this.pairs = pairs;
        }
        public List<Pair> getPairs() {
            return pairs;
        }
        @Override
        public Object accept(Visitor visitor) { return visitor.visit( e: this); }
    }
```

Grammar Example

```
grammar JSON;

// GRAMMAR RULES

json
    : dict
    ;

dict
    : '{' pair (',' pair)* '}'
    | '{' '}'
    ;

pair
    : '"' STRING '"' ':' value
    ;

value
    : str
    | num
    | dict
    ;

str
    : '"' STRING '"'
    ;

num
    : NUMBER
    ;

// LEXER TOKENS

STRING
    : (ESC | SAFECODEPOINT)*
    ;

fragment ESC
    : '\\' (["\\/bfnrt] | UNICODE)
    ;

fragment UNICODE
    : 'u' HEX HEX HEX HEX
    ;

fragment HEX
    : [0-9a-fA-F]
    ;

fragment SAFECODEPOINT
    : ~ ["\\\u0000-\u001F]
    ;

NUMBER
    : INT
    ;

fragment INT
    : '0' | [1-9] [0-9]*
    ;

// \- since - means "range" inside [...]
WS
    : [ \t\n\r] + -> skip
    ;
```

Adding Actions

```
grammar JSON;
// GRAMMAR RULES
Json returns [Dict ast] :
   : d = dict { $ast = $d.ast; } // return the ast parsed for the dictionary
   ;

dict returns [Dict ast]
   locals [ArrayList<Pair> list]
   @init {$list = new ArrayList<Pairs>(); }
   : '{' p=pair { $list.add($p.ast); }
         (',' p=pair {$list.add($p.ast);} )*
     '}' { $ast = new Dict($list); }
   | '{' '}' { $ast = new Dict($list); }
   ;
pair returns [Pair ast]
   : s='"' STRING '"' ':' v=value { $ast = new Pair($s.text, $v.ast); }
   ;

   ;
value returns [JsonTree ast]
   : s=str    { $ast = $s.ast; }
   | n=num    { $ast = $n.ast; }
   | d=dict      { $ast = $d.ast; }

   ;
str returns [Str ast]
   : s='"' STRING '"' { $ast = new Str($s.text); }
   ;
num returns [Num ast]
   : n=NUMBER { $ast = new Num(Integer.valueOf($n.text)); }
   ;
```

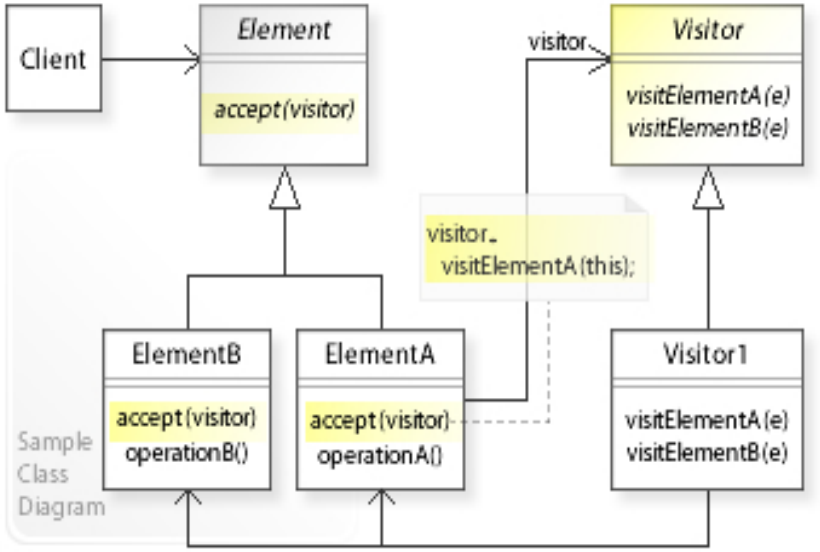
Grammar Rules Components

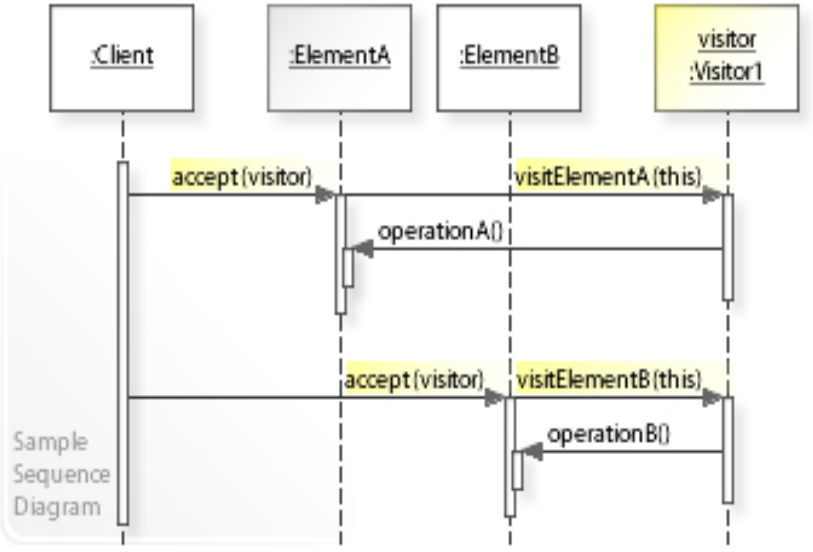
- Return type
  - dict returns [Dict ast]
- We can define local variables
  - locals [ArrayList<Pair> list]
- We have to initialize them
  - @init {$list = new ArrayList<Pairs>(); }
- An element of the expression can have an action
  - p=pair { $list.add($p.ast); }
- Each rule can have a final action
  - { $ast = new Dict($list); }

Visitor Pattern

- Object-oriented design pattern
- Allow to add new operations to existent object structures without modifying the structures
- A client accepts a visitor and a visitor traverses the object structure calling *accept(visitor)*
- *The visitor performs operations on the element*

Visitor Pattern





Visitor Example

- Define a simple data structure representing a Json tree
- We support strings, numbers (integers) and Json objects
- We will implement the Json tree to support the visitor pattern

Visitor Example

```
public abstract class JsonTree {

    public abstract Object accept(Visitor visitor);

    public static class Str extends JsonTree {
        String str;
        public Str(String str) { this.str = str; }
        public String getStr() {
            return str;
        }
        @Override
        public Object accept(Visitor visitor) { return visitor.visit( e: this); }
    }

    public static class Num extends JsonTree {
        int num;
        public Num(int num) { this.num = num; }
        public int getNum() { return num; }
        @Override
        public Object accept(Visitor visitor) { return visitor.visit( e: this); }
    }

    public static class Dict extends JsonTree {
        List<Pair> pairs;
        public Dict(List<Pair> pairs) {
            this.pairs = pairs;
        }
        public List<Pair> getPairs() {
            return pairs;
        }
        @Override
        public Object accept(Visitor visitor) { return visitor.visit( e: this); }
    }
```

Visitor Example

```
public interface Visitor<T> {
    public T visit(Str e);
    public T visit(Num e);
    public T visit(Dict e);
}
```

Visitor Example

- Implement a visitor to return a string representation of the json object
- The visitor StringifyVisitor implements all the visit methods defined in the interface
- The value it returns is a string

Visitor Example

```java
public class StringifyVisitor implements JsonTree.Visitor<String> {
    @Override
    public String visit(JsonTree.Str e) {
        return "\"" + e.getStr() + "\"";
    }

    @Override
    public String visit(JsonTree.Num e) {
        return "" + e.getNum();
    }

    @Override
    public String visit(JsonTree.Dict e) {
        StringBuilder tokens = new StringBuilder("{");
        for (int i = 0; i < e.getPairs().size(); i++) {
            JsonTree.Pair pair = e.getPairs().get(i);
            tokens.append("\"" + pair.getKey() + "\"")
                    .append(":")
                    .append(pair.getValue().accept( visitor: this));
            if (i < e.getPairs().size() - 1) {
                tokens.append(", ");
            }
        }

        return tokens.append("}").toString();
    }
}
```