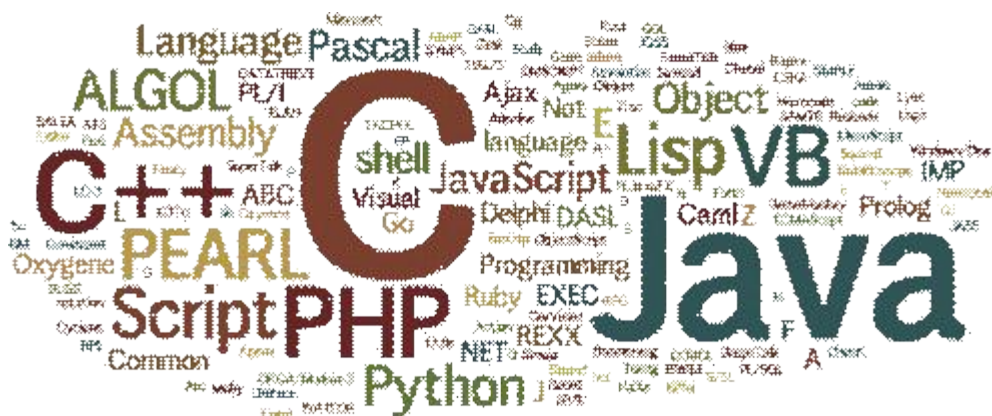


프로그래밍 언어 개론 9주차



02분반

201402421 정민우

1. 문제해결 방법

과제 : 파일입력,파일출력에서 콘솔 입/출력으로 바꾸기

작동 : 프로그램이 실행되면 console 창에서 interpreter 형식으로 명령어 입력을 받고, 그 아래에 처리한 결과값을 보여준다.

구현 방식 :

```
public class CuteInterpreter {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        Scanner sc = new Scanner(System.in);  
        String _string;  
  
        while(true) { //반복적으로 입력을 받는다.  
            System.out.print("> ");  
            _string = sc.nextLine(); //입력받아서 저장  
            if(_string.contentEquals("exit")) break; //프로그램 종료조건(임의로 exit으로 만들었다.)  
            CuteParser cuteParser = new CuteParser(_string);  
            //cuteParser 객체 생성(매개변수로 File타입에서 String타입으로 바꾼것이 중요)  
            CuteInterpreter interpreter = new CuteInterpreter(); //여기부터는 바꿀필요 없다.  
            Node parseTree = cuteParser.parseExpr();  
            Node resultNode = interpreter.runExpr(parseTree);  
            NodePrinter nodePrinter = new NodePrinter(resultNode);  
            nodePrinter.prettyPrint(); //prettyPrint부분을 콘솔 출력으로 수정했다.  
        }  
        System.out.println("END PROGRAM");  
    }  
}
```

우선 Main부분이다. 더 이상 ParserMain은 필요가 없다. 파일의 입/출력이 필요 없이 단순히 콘솔 입/출력으로 바꾸었기 때문이다. 그리고 Main에서 달라진 부분은 파일을 읽는부분을 삭제한것과, CuteParser 객체 생성시 인자로 String타입 변수를 넣어준 것이다. 이 말은 CuteParser의 생성자도 바뀌었다는 의미이다. 이제 CuteParser 클래스를 보겠다.

```

public class CuteParser {
    private Iterator<Token> tokens;
    private static Node END_OF_LIST = new Node() {};
    public CuteParser(String string) { //바뀐 생성자부분
        try {
            tokens = Scanner.scan(string);
            //Scanner클래스의 scan부분도 file입력이었는데 string으로 바꾸었다.
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}

```

보게되면 원래는 `tokens = Scanner.scan(file)` 이런식으로 `scan`함수의 인자가 `File`형식이었지만, `string`으로 바꾸었다. 그렇다면 다시 `Scanner`클래스의 `scan`함수를 보자.

```

public class Scanner {
    public static Iterator<Token> scan(String string) throws FileNotFoundException {
        ScanContext context = new ScanContext(string);
        //ScanContext객체를 생성한다. (입력한 string값이름)
        return new TokenIterator(context); //토큰반복자 객체 생성
    }

    public static Stream<Token> stream(String string) throws FileNotFoundException {
        Iterator<Token> tokens = scan(string); //토큰 반복자 배열에 String값을 기반으로 Tokenize한다.
        return StreamSupport.stream(
            Spliterators.spliteratorUnknownSize(tokens, Spliterator.ORDERED), false);
    }
}

```

여기가 `lexer`패키지의 `Scanner`클래스내부의 `scan`함수이다. 보게되면 원래 `ScanContext`의 `context`에 객체를 저장할 때, 생성하는 객체에 인자가 `file`이었지만, `String`타입으로 바꾸었음을 알 수 있다. 그렇다면 다시 `ScanContext` 클래스를 보자

```

class ScanContext {
    private final CharStream input;
    private StringBuilder builder;

    // ScanContext(File file) throws FileNotFoundException {
    //     this.input = CharStream.from(file);
    //     this.builder = new StringBuilder();
    // }

    ScanContext(String string) throws FileNotFoundException {
        this.input = CharStream.from(string);
        //CharStream.from를 통해 CharStream를 불러온다.
        this.builder = new StringBuilder();
    }
}

```

여기서도 바뀐 점이 있다. CharStream 클래스에서 from 함수를 호출할 때 불러오는 인자가 원래는 File 타입이었지만 String 형식으로 바꾸어서 input값을 바꾸도록했다.

마지막으로 CharStream 클래스를 보자

```

class CharStream { //CharStream
    private final Reader reader;
    private Character cache;

    // static CharStream from(File file) throws FileNotFoundException {
    //     return new CharStream(new FileReader(file));
    // }

    static CharStream from(String string) throws FileNotFoundException {
        return new CharStream(new StringReader(string));
        //StringReader 객체를 Reader로 저장
    }

    CharStream(Reader reader) { //생성자
        this.reader = reader;
        this.cache = null;
    }
}

```

CharStream클래스의 from 함수이다. 원래는 File을 인자로 가져오고, Reader값을 FileReader 타입의 객체로 저장했었지만, 이제는 String값을 인자로 가져오고, Reader 값을 StringReader객체로 저장하게했다. 그래서 이제는 파일을 읽는 것이 아니라, 들어온 String값을 읽어서 처리한다.

```

public void prettyPrint(){
    printNode(root); //sb값에 결과값 저장
    System.out.println("..." + sb.toString()); //console창에 결과값 sb출력

    try(FileWriter fw = new FileWriter(OUTPUT_FILENAME);
        PrintWriter pw = new PrintWriter(fw)){
        pw.write(sb.toString());
    }catch (IOException e){
        e.printStackTrace();
    }
}

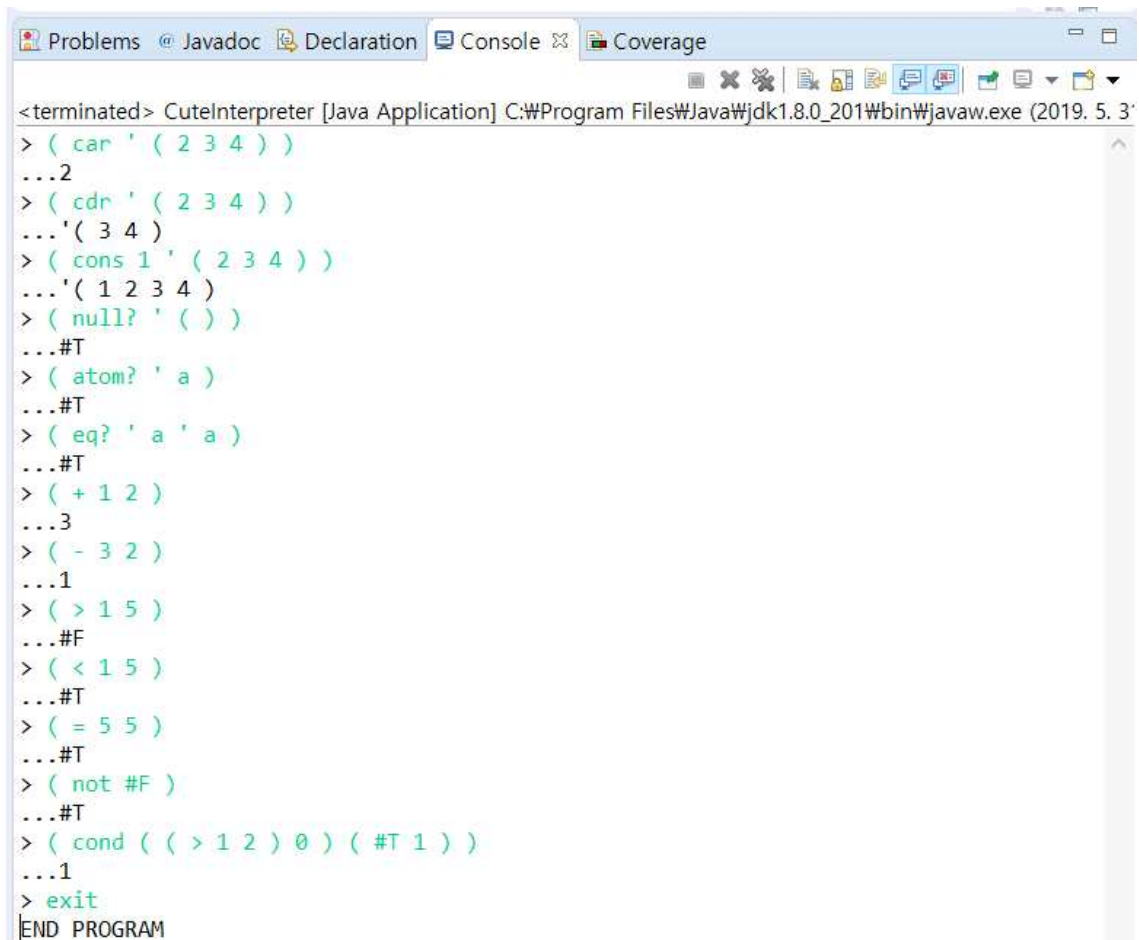
```

처리를 끝내고, prettyPrint() 함수를 호출해서 출력을 진행한다. 기존에는 outputFile를 만들어서 거기에 결과값을 넣는 방식이었지만(주석부분) 이제는 모든 처리가 끝난 결과값이 sb라는 변수에 저장되어있으므로, 그걸그냥 console창에 출력하기만하면 돼서 System.out.println() 함수로 출력하는방식으로 단순하게 바꾸었다.

2. 느낀점

이번 과제는 입/출력 방식부분만 바꾸는것이기 때문에, 코드를 따라가면서 File Reading부분을 String Reading으로 바꾸기 만해도 끝나서 지금까지의 과제보다는 쉬운편이었다. 그렇지만, 여태까지는 입/출력부분은 잘보지 않았는데 이번 과제를 통해서 그 과정들을 보면서 새로운 부분을 이해할 수 있게 되었다.

3. 테스트 코드 실행 결과



```
<terminated> CuteInterpreter [Java Application] C:\Program Files\Java\jdk1.8.0_201\bin\javaw.exe (2019. 5. 3)
> ( car ' ( 2 3 4 ) )
...2
> ( cdr ' ( 2 3 4 ) )
...'( 3 4 )
> ( cons 1 ' ( 2 3 4 ) )
...'( 1 2 3 4 )
> ( null? ' ( ) )
...#T
> ( atom? ' a )
...#T
> ( eq? ' a ' a )
...#T
> ( + 1 2 )
...3
> ( - 3 2 )
...1
> ( > 1 5 )
...#F
> ( < 1 5 )
...#T
> ( = 5 5 )
...#T
> ( not #F )
...#T
> ( cond ( ( > 1 2 ) 0 ) ( #T 1 ) )
...1
> exit
END PROGRAM
```

8주차 과제 PPT에 있는 입력들을 몇 개 넣어 보았다. 모두 정상적인 결과로 연속적인 출력이 되는 것을 확인했다. 그리고 마지막으로 내가 추가한 exit을 입력하자, 반복문을 빠져나와 입력을 종료하는것까지 확인할 수 있었다.

* Item 2,3 과제 조원

201402421 정민우(본인)

201502129 홍석우

201602086 최유정