TaML

Table Manipulation Language

Adam Dossa (aid2112)
Qiuzi Shangguan (qs2130)
Maria Taku (mat2185)
Le Chang (lc2879)

Columbia University

19th December 2012

Overview

1. Simple C-like language for building, editing, and manipulating tables/ spreadsheets.

2. Built-in type: Table, Line, Cell and others.

3. static typed.

Motivation

1. Quickly and efficiently manage budgets, calculate yearly taxes.

2. Perform various mathematical calculation. Keep track of various types of numerical data and the relationships between this data.

3 Show visual result in a table and play games on a table.

Language Tutorial

- Tables are always distinct – they have their own memory allocation.
- Cells and lines are references to cells in a table
- The carat operator ^ allows us to access the values of cells, rather than the cell itself.

Distinct Memory Allocations

```
table tab1 = ([10,5],int);
table tab2 = tab1[1~5, @];
table tab 3 = tab1[@,@];
```

References to other Cells

```
line line1 = tab1[0, 0~5];
cell cell1 = tab1[1,1];
^cell1 = 50; →   also changes
                value of tab1[1,1]
```

Language Tutorial (example)

```
string good = "your budget is good";
string bad = "spending too much money";
table t = ([10,10],float);
cell expenses = t[0,1];
cell maxBudget = t[1,1];

func void main(){
        setBudget(999.99);
        fillBudget();
        checkBudget();
}

func void setBudget(float maxBud){
        ^maxBudget = maxBud;
}
```

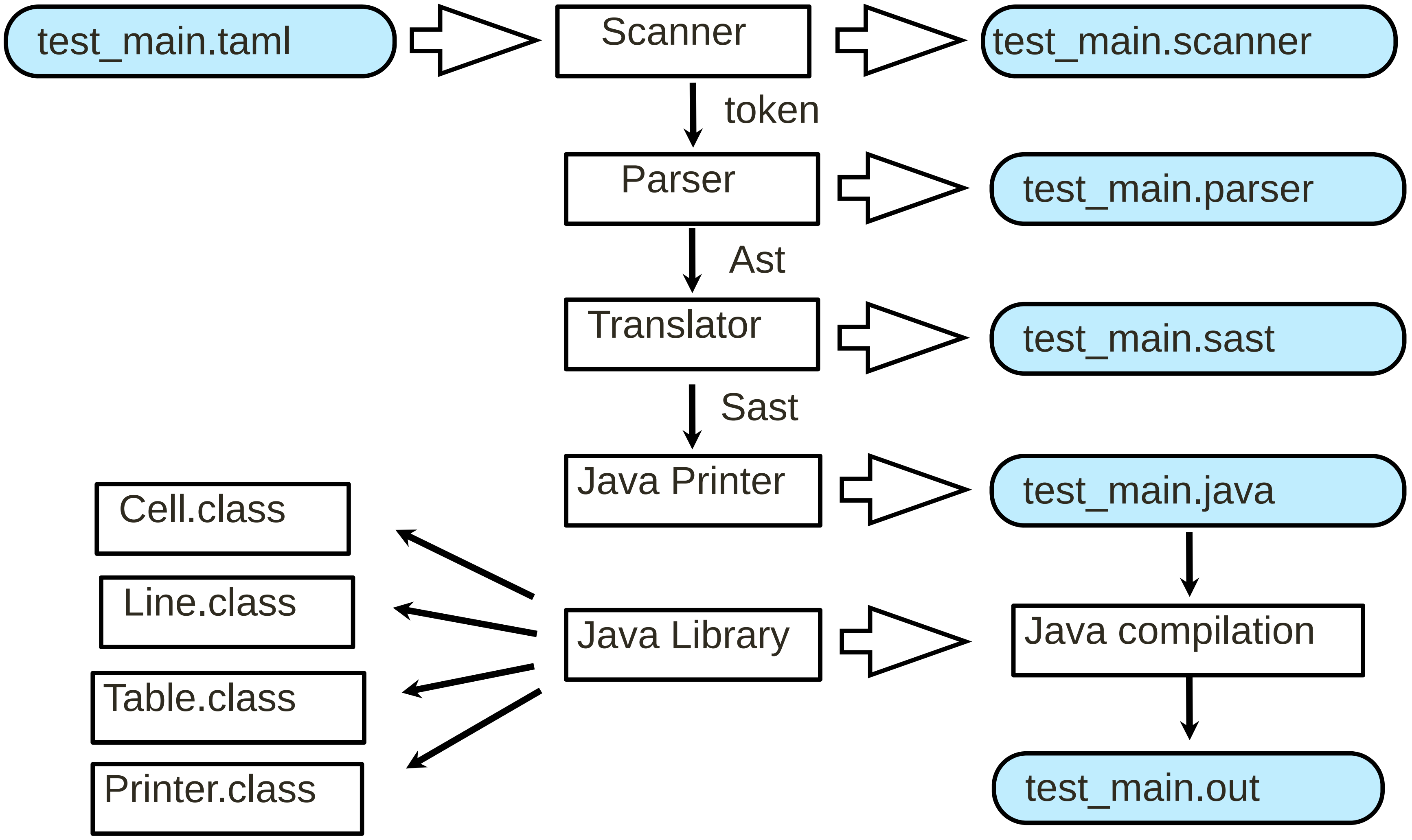
```
func void fillBudget(){
    ^expenses = 0.0;
    int i;
    for(i=0; i<10; i=i+1){
        ^t[i,0] = 100.0;
        ^expenses = ^expenses + ^t[i,0];
    }
}

func void checkBudget(){
     if(^expenses > ^maxBudget){
          print(bad);
     } else {
          print(good);
     }
}
```

Implementation

- **Development Procedures**
- Scanner,Testing
- Parser,Testing
- Semantic Checking,Testing
- Java-printer,Testing
- Java_lib,Testing
- **Development Tools**
- Ocamlyacc Ocamllex
- Command-line with Makefile for Ocaml
- Eclipse, Emac, Vi for Java,Ocaml editing
- Git and Github for version control and repository

Implementation



Implementation

Class Cell<E>
----
private E value
----
void Cell()
E getVal()
void setVal()
public print()

Class Line<E>
----
private Cell<E>[] line
private int lineLength
----
void Line()
assignLine(Table, int, int, int, int)
assignLine(Table, String, String, int, int)
assignLine(Table, int, int, String, String)
createLinecopy(int, int)
createLinecopy(String, String)
E getCellValue(int)
void setVal(int, E)
Cell<E> getCell(int)
void print()

Class Table<E>
----
private Cell<E>[][] table
private int numRows
private int numColumns
----
void Table()
Table<E> createTableCopy(int,int,int,int)
Table<E> createTableCopy(String,String,int,int)
Table<E> createTableCopy(int,int,String,String)
Table<E>
createTableCopy(String,String,String,String)
E getCellValue(int,int)
void setVal(int,int,E)
void setVal(E)
void setVal(int,int,int,int,E)
void setVal(String,String,int,int,E)
void setVal(int,int, String,String,E)
void print()

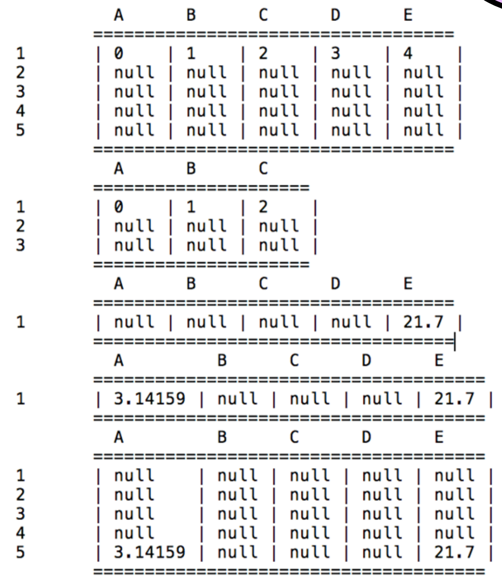
.taml

```
func void main(){      table intTable = ([5,5], int);
= 0;                    ^intTable[0,1] = 1;   ^intTable[                    [0,3]
= 3;                    ^intTable[0,4] = 4;   print(intTable);        table
smallIntTable = intTable[0~2, 0~2];           print(smallIntTable); table
floatTable = ([5,5], float);                  ^floatTable[4,4] = 21.7;
                        line floatLine = floatTable[4,@];
                        print(floatLine);                             ^floatLine[0] =
3.14159;                                      print(floatLine);
                        print(floatTable);    line smallFloatLine =
floatLine[0~2];         ^smallFloatLine[1] = 1.111;
                        print(smallFloatLine);
                        print(floatLine);
........
```

```java
public class test_main
{
public static void main(String[] args)
{
Table<Integer> intTable = new Table<Integer>(5,5);
intTable.setVal(0,0,0);
intTable.setVal(0,1,1);
intTable.setVal(0,2,2);
intTable.setVal(0,3,3);
intTable.setVal(0,4,4);
Printers.print(intTable);
Table smallIntTable=intTable.createTableCopy(0,2,0,2);
Printers.print(smallIntTable);
Table<Float> floatTable = new Table<Float>(5,5);
floatTable.setVal(4,4,21.7f);
Line<Float> floatLine= new Line<Float>();
floatLine.assignLine(floatTable,4,4,"ALL","ALL");
Printers.print(floatLine);
floatLine.setVal(0,3.14159f);
Printers.print(floatLine);
Printers.print(floatTable);
Line<Float> smallFloatLine=floatLine.createLineCopy(0,2);
smallFloatLine.setVal(1,1.111f);
Printers.print(smallFloatLine);
Printers.print(floatLine);
........
```

.java

.out



```
           A      B      C      D      E
         ====================================
1        | 0    | 1    | 2    | 3    | 4    |
2        | null | null | null | null | null |
3        | null | null | null | null | null |
4        | null | null | null | null | null |
5        | null | null | null | null | null |
         ====================================
           A      B      C
         =====================
1        | 0    | 1    | 2    |
2        | null | null | null |
3        | null | null | null |
         =====================
           A      B      C      D      E
         ====================================
1        | null | null | null | null | 21.7 |
         ====================================
           A         B      C      D      E
         =======================================
1        | 3.14159 | null | null | null | 21.7 |
         =======================================
           A         B      C      D      E
         =======================================
1        | null    | null | null | null | null |
2        | null    | null | null | null | null |
3        | null    | null | null | null | null |
4        | null    | null | null | null | null |
5        | 3.14159 | null | null | null | 21.7 |
         =======================================
```

Summary

- The team worked well together - despite the pressure towards the ends we never descended into violence / anger / bickering / finger pointing.
- TaML was a complex language choice disguised as a simple language choice.
- Working language delivered, albeit with some known limitations and idiosyncrasies.
- Learning a new language is hard, learning a new language and using it to build a translator is very hard.

Lessons Learnt

1) Languages with dynamic / generic types are hard!

2) Adapting process to fit team dynamic makes everyone more productive.

2) Having modular test cases lets you pick up bugs earlier, leading to less complex debugging later.

3) Following the standard approach (scanner / parser / ast/ sast / printer) makes sense – trying to skip steps doesn’t.

• The project was a steep learning curve in two dimensions (Ocaml / Translators).

8) Defining / limiting scope a necessary part of working to a deadline.