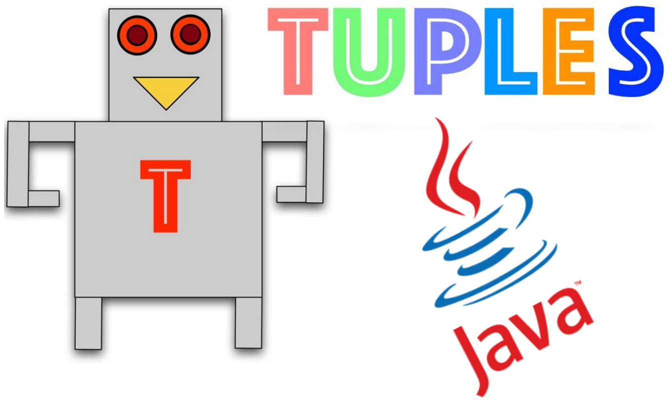
Fall 2016 COMS4115

Programming Languages and Translators

Project Proposal

Professor Stephen Edwards
Columbia University

Due September 28, 2016
at 4:10pm



Java +-

(Java but better but also worse)

Team Members

Anna Wen (aw2802) - Tester

Ashley Daguanno (ad3079) - Manager

Zeynep Ejder (ze2113) - Language Guru

Amel Abid (aa3454) - Systems Architect

Tin Nilar Hlaing (th2520) - Systems Architect

Project Overview + Motivation

Java +- is a programming language largely based on Java that will have the added data types *tuple* and *lightweight*. Our language will compile into C.

Why tuples?

> *"Tuples are useful for a variety of reasons. If you need to package several objects together, but don't want to define a class just for that purpose, you can use a tuple."* - Gilad Bracha, https://gbracha.blogspot.com/2007/02/tuples.html
>
> *"Tuples are useful because they can let you do things like quickly return multiple values from a function, easily group related items together, and so forth."* - Reddit user michael0x2a

The keyword *lightweight* refers to the lightweight object associated with every class object. This lightweight allows for access to an object's properties without the burden of also carrying all of its methods. This will make working with an object's data cheaper, since we have already extrapolated that object's data into more lightweight object.

Syntax

Data Types

- number - All number types will automatically be decimals. No ints, floats, longs, etc.
- String - Enclose in quotes, i.e. “Hello World”
- Boolean - True or False
- Tuple - A sequence of objects enclosed in parentheses, i.e. (“Hi”, 2)
- Array - Standard Java arrays with 0-based indexing, i.e. [24,52,60]
- Class Objects - User defined objects.
- lightweight - The lightweight counterpart to full class objects.

Object Definition and Object Features

Our object oriented language will have two types of objects:

- Regular class objects with instance variables and methods
- Lightweight Object - holds the properties(instance variables) of an object and allows for (only) access of these properties
  - Example of an instance of a Person class with instance variables name and age: [(“name”, “John”);(“age”,6)]
  - The user will get this lightweight object by calling .lw() on the regular object. They will not need to define a method called lw() this will be an inherited method that every Class will have. If the user wants the instance variable returned in a certain way, they can add a getter method that the lw() method will use to create the lightweight object.
  - Any changes made to an object’s properties will not change the values stored in the lightweight object if it was created before those modifications.

Keywords

- public - visible everywhere
- private - visible in lightweight objects
- hidden - not visible in lightweight objects

Complete Statements

- Semicolons will end statements, ala ;

Indentation

- Indentation does not matter - all that matters is that individual statements and the overall class structure are syntactically correct

Comments

- Standard Java syntax
    - // for single line comments
    - /* */ for block comments

Print Statements

- Print()
    - Ex: Print("Hello World!");

Basic Operators

- Comparison Operators: ==, <=, >=, !=, <, >
- Mathematical Operators: +, -, *, /
- Variable Assignment: =
- Logical Operators: AND, OR
- String Concatenation: +

Looping Statements

- Standard Java for and while loops. No use of i++, must specify i=i+1.

Control Flows

- Standard Java if/else if/else syntax, though brackets are necessary.
- No switch statements.

Bracket Usage:

- Standard Java bracket usage - use to enclose classes, methods, etc.

Examples

GCD Algorithm:

```
Class Test {

      public int GCD(number a, number b) {
            for(number i = a; i <= b; i = i+1) {
                  if (a > b) {
                        a = a-b;
                  }
                  else {
                        b = b-a;
                  }
            }
      }
      public void main() {
            number a = 42;
            number b = 80;
            number result = GCD(a,b);
            Print(result);
      }

}
```

Array Example:

```
Class TestArrays {

      public String greetStudents() {
            String[] myStudents = [“John”, “Abigail”, “Stephen”, “Thomas”];
            for(number i = 0; 0 < myStudents.length; i=i+1) {
                  Print(“Hi” + myStudents[i] + “!\n”);
            }
      }

      public void main() {
            greetStudents();
      }
}
```

Returning Tuples:

```
Class UseTuples {

      public tuple<"String", "String", "String"> getValues() {
            return ("Tommy", "Jones", "1234 Cherry Lane.");
      }

      public void main() {
            tuple<”String”, “String”, “String”> info = getValues();
            Print(“First name = “ + info[0]);
            Print(“Last name = “ + info[1]);
            Print(“Address = “ + info[2]);
      }
}
```

Working with lightweights:

```
Class Car {
    private String brand;
    private String color;
    private number size;
    hidden  number serialNum;

    public car(brand, color, size, serialNum) {
        this.brand = brand;
        this.color = color;
        this.size  = size;
        this.serialNum = serialNum;
    }

    public String getBrand() {
        return this.brand;
    }

    public String getColor() {
        return color;
    }

    public number getSize() {
        return size;
    }

    public number getSerialNum() {
        return serialNum;
    }

    public number speed() {
        //some speed method
    }
}

Class DummyClass {
    public void main() {
        Car newCar = new Car(“blue”, 5);
        lightweight lwCar = newCar.lw();
        // lwCar = (brand: “Toyota”, color: “blue”, size: 5)
        // Note the absence of instance variable serialNum due to it
           being hidden
        String brand = lwCar[brand];
        Print(“The brand of this car is” + brand);
```

```
          }
  }
```