**Trabajo Práctico Especial**

# 1. Introducción

Cuando hablamos de **Java**, en realidad nos estamos refiriendo a dos cosas a la vez. Por un lado, es un lenguaje de programación, orientado a objetos, fuertemente tipado, que se compila a un código objeto ejecutable que detalla instrucciones para un procesador virtual, la Java Virtual Machine. Por otro lado, es un framework, una plataforma, con una enorme cantidad de librerías, funcionalidades y herramientas. Una de las características que han hecho de Java el segundo lenguaje más utilizado (ver TIOBE index) es justamente el uso de la JVM como plataforma donde corren los programas Java y la abstracción al sistema operativo: java es multiplataforma, y un mismo Binario Java puede correr en cualquier sistema.

## 1.1. Objetivo

El objetivo del presente es construir un Compilador de un lenguaje procedural simple, y que el programa objeto generado pueda ser justamente ejecutado en una JVM, Java Virtual Machine. El primer paso será definir la gramática de este lenguaje , y a partir de la gramática, construir un compilador que genere como salida un programa objeto que pueda correr como una aplicación Java, dentro de una JVM.

# 2. Lenguaje

La primera parte es la definición del lenguaje que el Compilador aceptará.
El mismo debe contener:

- Tipos de datos: numérico y cadenas.
- Constantes, Delimitadores.
- Operadores aritméticos, relacionales, lógicos, de asignación.
- Bloque Condicional (if).
- Bloque Do-While (repetición hasta condición).
- Un bloque principal de ejecución.
- Proveer un mecanismo de entrada de datos y de salida.
- El Compilador debe generar código para ser ejecutado en una JVM, Java Virtual Machine (cualquier versión).
- Debe ser diferente de Java (por obvias razones).

Ejemplo de un lenguaje posible:

```
int x;
int y;
main() {
  x = 6;
  y = 1;
  while (x>0) {
   y=y*x;
   x=x-1;
   }
  puts("El factorial de 6 es: ");
  puts(y);
```

```
  puts("\n");
  puts("El valor de x+1 es:");
  puts(x+1);
}
```

(el anterior es un ejemplo, pueden definir ustedes las palabras reservadas que deseen, y la estructura que les parezca más conveniente).

**Piensen creativamente una idea detrás del lenguaje. Algo que surja en base a su propia experiencia de programación y una idea de funcionamiento básica detrás de la creación del lenguaje.**

# 3. Yet Another Compiler Compiler

Los compiladores de compiladores, son herramientas que permiten en base a una definición de un autómata finito y una gramática en BNF desarrollar los dos componentes principales de un compilador: el analizador léxico y el analizador sintáctico.

Para este trabajo, orientado a Java, usaremos las librerías de JFlex para el *scanner*, y Cup para el *parser*

## 3.1. CUP y jFlex

La estructura de los archivos de gramáticas de CUP es prácticamente igual al famoso motor YACC con las salvedades del lenguaje Java.

La librería tiene dos jars: Uno, es el compilador propio del archivo .cup que contiene la definición de la gramática ( http://www2.cs.tum.edu/projects/cup/java-cup-11a.jar ). La librería tiene un jar adicional que se usa en tiempo de ejecución del compilador ( http://www2.cs.tum.edu/projects/cup/java-cup-11a-runtime.jar ).

Los pasos para ejecutar son:

```
java -jar java-cup-11a.jar Parser.cup
```

Genera como salida el archivo **Parser.java** y **sym.java** (el analizador sintáctico y la tabla de símbolos).

```
java -jar JFlex.jar Scanner.jflex
```

Genera **Scanner.java**, el analizador léxico.

Luego compilan todo, y generan su propio compilador *Compilador.java*

```
javac -classpath java-cup-11a.jar Scanner.java sym.java Parser.java
```

En el ejemplo **JavaCompiler.zip** se encuentra un ejemplo completo de utilización.

# 4. Java Assembler

El compilador que ustedes generan, va a recibir como entrada un programa con código en el lenguaje que ustedes tienen que crear, y cómo salida tiene que generar código que sea ejecutable en una Java Virtual Machine.

Típicamente, este código, puede ser Assembler de la JVM. La JVM es después de todo una *máquina* virtual que puede ejecutar sentencias en un assembler virtual que es el famoso **bytecode**.

Una representación, con símbolos legibles, del Bytecode es como sigue:

```
.class public HelloWorldPackage.HelloWorld
.super java/lang/Object

.method public <init>()V
   aload_0
   invokenonvirtual java/lang/Object/<init>()V
   return
.end method

.method public static main([Ljava/lang/String;)V
   .limit stack 2
   .limit locals 2
   getstatic      java/lang/System/out Ljava/io/PrintStream;
   ldc            "Hello World."
   invokevirtual  java/io/PrintStream/println(Ljava/lang/String;)V
   return
.end method
```

El projecto Jasmine, la versión 2.4, es una vieja y poca mantenida herramienta que permitía en base a código assembler de bytecode legible, como el anterior, generar código assembler binario puro para poder ser ejecutado en las JVMs.

Pueden bajarse de la página del proyecto Jasmine, la versión 2.4. Además de muchas librerías y funcionalides adicionales, la descarga contiene el ejecutable java **jasmin.jar** que lo pueden usar para convertir código Jasmine en Bytecode Java.

Si al programa de ejemplo anterior, lo graban en el archivo "HelloWorldPackage.HelloWorld", pueden hacer

```
java -jar jasmin.jar HelloWorldPackage.HelloWorld
```

Esto compila el código assembler en Bytecode que luego pueden ejecutar en una JVM haciendo:

```
java -cp . HelloWorldPackage.HelloWorld
```

(Recuerden los ajustes de paths necesarios para que funcionen estos ejemplos).

# 5. Bytecode

El famoso bytecode de Java es una estructura de árbol, representando el árbol sintáctico decorado de la clase java con la que fue generado. El árbol contiene diferentes tipos de nodos en una estructura predefinida y generalmente en las correspondinetes hojas aparecen las operaciones mínimas en assembler de la jvm. Estos son operandos en assembler de un solo byte (por eso bytecode). Aproximadamente 3/4 de los códigos posibles de un byte ya se encuentran en uso en las últimas versiones de Java.

# 6. ASM

Como Jasmine tiene muchos problemas de falta de mantenimiento, obsolescencia y poca documentación, otra forma es utilizar el proyecto ASM. Esta librería permite generar y manipular on runtime el código bytecode como una colección de objetos vinculados por el propio árbol representado en el bytecode. Si bien la herramienta esta pensada para la modificación de clases existentes, también permite generar código ejecutable en una JVM y solucionar muchos de los detalles de la generación de código simplemente utilizando métodos y funciones ya armadas.

# 7. Material a entregar

Deben Entregar:

- Proyecto Java Completo: código fuente, librerías adicionales, maven, ant o mecanismo de compilación similar.
- Los ejecutables del compilador (como .jars). Pueden incluir herramientas y librerías adicionales que ustedes utilicen.
- Un README.md explicando cómo compilar y ejecutar el programa.
- Cinco(5) programas de ejemplo en el lenguaje: factorial, test de primalidad, fibonacci, tateti, rock paper scissors
- Dos(2) programas adicionales de muestra inventados por ustedes.
- Un informe que contenga, en este orden:
  - Carátula
  - Índice
  - Consideraciones realizadas (no previstas en el enunciado).
  - Descripción del desarrollo del TP.
  - Descripción de la gramática.
  - Dificultades encontradas en el desarrollo del TP.
  - Futuras extensiones, con una breve descripción de la complejidad de cada una.
  - Referencias.

El informe lo pueden escribir en inglés.
Adicionales deseables que se evaluarán positivamente:

- Optimizaciones.
- Benchmarking de los tiempos de ejecución de los cinco programas de ejemplo generados por el Compilador vs. programas similares en Java (u otro lenguaje de scripting que corra en la JVM, como Scala).
- Posibilidad de utilizar en el lenguaje generado librerías de la extensa biblioteca de Java (por ejemplo abrir un socket, o implementar un servicio REST).
- Agregados extra al lenguaje como posibilidad de recursión, tipos de datos de punto flotante, manejo de listas y colas, etc.

**Importante:** No hay ningún inconveniente en utilizar librerías públicas, soluciones similares públicas, soluciones de foros, etc., pero es necesario aclarar, y enumerar cada una de ellas en la sección Referencias. No se aceptan bloques de código públicos implementados verbatim sin ningún tipo de análisis. Tampoco implementaciones que resuelven problemas que no están detallados (e.g. implementa un garbage collector sin explicar cómo). Tampoco hay inconveniente en que interactuen con otros grupos. *La diferencia entre plagio y ciencia es una referencia.*

## 8. Grupos

Los grupos serán los definidos al inicio de la cursada.

## 9. Fecha de entrega

El material a entregar puede ser o bien enviado por email a `rramele@gmail.com` en una carpeta ZIP rotulada con el nombre del grupo, o bien en un Branch o Tag de Git, Bitbucket u otro SCM similar. En el segundo caso, es requerido que envíen un email detallando el nombre del repositorio y el nombre del TAG que corresponde a la entrega.

El TP se debe entregar antes de 03/07/2016 23:59 GMT -3.

# 10. Criterio de Corrección

- A. Informe: secciones, prosa, errores ortográficos, claridad en la exposición.
- B. Cumplimiento de consignas.
- C. Implementación del TP: investigación, creatividad, innovación, practicidad , definición y exposición de la idea subyacente, alineamiento con los temas vistos en la materia.
- D. Calidad en la presentación: armado del proyecto, documentación del código, estructura del código, scripts de automatización, test de regresión, etc.
- E. $+\alpha$: Puntos extras agregados más allá de los requerimientos del enunciado e incluso del trabajo mismo.

Cada item es puntuado del 1-10 y la nota del TP surge del promedio de todos los puntos.

# 11. Material de consulta


- "The Java Virtual Machine" (Addison-Wesley) Lindholm et al, 2014
- "Compiladores, Principios, Técnicas y Herramientas",Aho, Sethi, Ullman, Addison Wesley.(El Libro del Dragón), capítulos 1,2 y 3.
- http://dinosaur.compilertools.net/lex/
- Jasmine http://jasmin.sourceforge.net/
- Proyecto Jamaica: http://judoscript.org/articles/jamaica.html
- JVM Spec: http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-6.html
- JFlex http://jflex.de/
- Cup http://www2.cs.tum.edu/projects/cup/
- Jasclipse - Jasmin Plugin for Eclipse
- Demo http://www2.cs.tum.edu/projects/cup/minimal.tar.gz
- ASM Library: http://asm.ow2.org
- Programming Languages Pragmatic, Scott, 2009
- Practica Foundations for Programming Languages, Harper,2013
- http://matt.might.net/articles/grammars-bnf-ebnf/