1° Entrega - Idea y grupo para el proyecto

TetraLepis - Traductor Python a Java

v0.1.0

Equipo

Nombre	Apellido	Legajo	E-mail
Federico Inti	Garcia Lauberer	61374	fedegarcia@itba.edu.ar
Gianfranco	Magliotti	61172	gmagliotti@itba.edu.ar
Felipe	Mindlin	62774	fmindlin@itba.edu.ar
Francisco	Sendot	62351	fsendot@itba.edu.ar

Repositorio

- https://github.com/felipemindlin/TP-TLA

Idea

La idea a desarrollar consiste en crear un traductor de código Python a Java. El traductor debe crear un programa que sea funcionalmente equivalente al programa inicial.

Esta idea nos resultó interesante puesto a que la sencillez de la sintaxis de Python permite la creación de programas rápidamente sin estar demasiado tiempo ajustando nuestros pensamientos a la sintaxis y/o funcionamiento del lenguaje, sin embargo el hecho de que sea un lenguaje interpretado tiene las desventajas de generalmente tener peor rendimiento que uno compilado, y que ciertos errores (cómo de sintaxis) no se detectan hasta la ejecución de la línea errónea.

El nombre *TetraLepis* se basa en el género de una especie de serpiente nativa de la isla de Java, en Indonesia.

Construcciones

El traductor desarrollado debería ofrecer las siguientes prestaciones y funcionalidades:

(I). Deberá poder traducir un script vacío.
(II). Deberá poder traducir un script con una clase definida en el mismo.
(III). Deberá poder traducir un script con múltiples clases definidas en el mismo.

(IV). Deberá poder traducir un script con clases que contengan inner classes o que posean herencia.

(V). Deberá poder traducir un script con sólo funciones definidas en el mismo.

(VI). Deberá poder traducir un script con funciones definidas, una o múltiples clases definidas en el mismo.

(VII). Deberá poder realizar un análisis léxico del script y obtener su equivalente para Java.

(VIII). Deberá poder traducir el sistema de importación (no se podrán cubrir los casos donde se importen librerías que no tengan su contraparte directa en Java).

(IX). Deberá poder traducir las operaciones aritméticas de Python a su equivalente en Java.

(X). Deberá poder traducir las sentencias simples de Python a sentencias de Java.

(XI). Deberá poder traducir las sentencias compuestas de Python a sentencias compuestas de Java.

(XII). Deberá poder traducir las funciones Built-In de Python a Java.

(XIII). Deberá poder traducir los tipos Built-In de Python a Java, con el añadido de que se debe poder traducir de un sistema de tipado dinámico y Hinted-Typed a uno de tipado estático y Strong-Typed. La traducción deberá fallar si el script de Python no cumple con los requerimientos necesarios para poder tipar las variables en un determinado programa.

(XIV). Deberá poder traducir las excepciones Built-In de Python a excepciones de Java.

Ejemplos

Un programa de Python con el siguiente input:

```
def suma(x, y):
   return x + y
```

Generaría como output el siguiente código válido de Java:

```
public class Main {
   public static void main(String[] args) {}

   public static int suma(int x, int y){
      return x + y;
   }
}
```

Notar que el traductor **puede** inferir los tipos de determinados tipos de variables, en este caso, en base a la operación que empleó, donde el tipo más básico o "atómico" posible para la función *suma* es el tipo *int*.

Si se quiere además utilizar la función *suma* dentro del script como en el siguiente ejemplo:

```
def suma(x, y):
   return x + y

suma(3.0,2)
```

El traductor devolverá el siguiente output:

```
public class Main {
   public static void main(String[] args) {
      suma(3.0,2);
   }

   public static float suma(float x, float y){
      return x + y;
   }
}
```

Notar que al haber escrito explícitamente un número real en el programa de Python, el traductor ha inferido correctamente su tipo para un programa de Java, en este caso sería del tipo *float*.

Otro caso de uso puede ser traducir una clase de Python a una de Java, dado el script de entrada:

```
class Animal
   def __init__(self,bark,name)
      self.bark = bark
      self.name = name
```

Notemos que es una clase que, si bien por los nombres podría tener una cierta semántica que permitiría poder inferir los tipos a partir de la lectura del mismo script, no hay una definición explícita de los mismos, por lo que el traductor optará por usar Generics para poder definir los mismos en la versión de Java. El resultado de traducir el script dará como output un archivo llamado *Animal.java*, que contendrá lo siguiente:

```
public class Animal<A,B> {
   private A bark;
   private B name;

   public static float Animal(final A bark, final B name){
      this.bark = bark;
      this.name = name;
   }

   public A getBark(){
      return bark;
   }

   public void setBark(final A bark){
      this.bark = bark;
   }

   public B getName(){
      return name;
   }

   public void setName(final B name){
      this.name = name;
   }
}
```

Notar que seguirá las prácticas estándar de Java, es decir, variables privadas, con sus respectivos Getters y Setters.

Un script de Python que desee utilizar colecciones, no podrá utilizar colecciones que contengan más de un tipo, es decir, dado el siguiente script:

```
variable = [“hola”, 2, 1.0, “cómo estamos”]
```

El traductor no podrá traducirlo, ya que las colecciones en Java deben ser de un sólo tipo únicamente (salvo casos especiales como Map, con su contraparte en Python que es el Diccionario).

Casos de uso

Se proponen los siguientes 10 casos iniciales de **aceptación**:

(I). Un programa simple sin clases.
(II). Un programa que cree una clase y una instancia de la misma.
(III). Un programa que cree un método y lo ejecute
(IV). Un programa que lance una excepción y la maneje.
(V). Un programa que haga uso de las operaciones matemáticas definidas de base en Python.
(VI). Un programa que cree una lista y haga operaciones que modifiquen la misma.
(VII). Un programa con clases, subclases e *inner classes* en donde se instancian las mismas
(VIII). Un programa que opere con los tipos de datos numéricos en Python (int, float, complex)
(IX). Un programa que utilice los objetos especiales `None` y `NotImplemented`
(X). Un programa que cree un objeto `dict` y lo imprima a pantalla

Además, los siguientes casos de prueba de **rechazo**:

(XI). Un programa malformado con errores de sintaxis de Python.
(XII). Un programa que asigne a una variable un tipo de dato incompatible* con el tipo de dato inferido de la declaración inicial
(XIII). Un programa que llame variables no inicializadas previamente
(XIV). Un programa que acceda a campos inexistentes de una clase
(XV). Un programa que acceda a campos ocultos para el scope de acceso** de una clase
(XVI). Un programa que utilice métodos, funciones o campos de librerías no implementadas
(XVII). Un programa que utilice colecciones que contengan elementos de más de un tipo, siendo este incompatible con Java

*****Es decir que si una variable se declara inicialmente como, por ejemplo, un número entero y luego se le asigna un string (no numérico) no se debería generar una traducción. Esto lo decidimos de esta manera ya que consideramos estos tipos de sobreescritura cómo un error del programador que lleva a código innecesariamente confuso
** Por ejemplo, si una función global intenta a acceder a un campo privado de una clase