

Universidad Del Valle De Guatemala

CC 3032-Construcción de Compiladores



Laboratorio 1

Roberto Javier Mombiela Herrera 20067

Jose Andrés Hernández Guerra 20053

Guatemala, 30 de julio del 2023

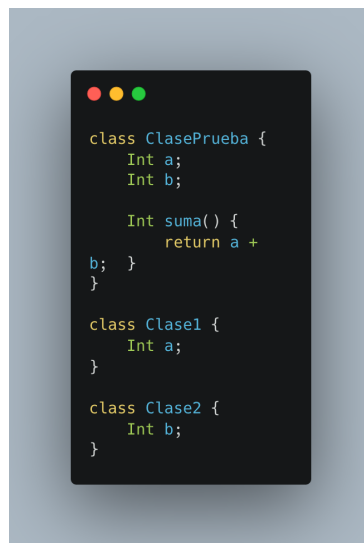
Diseño de sistema de tipos

Definiendo las reglas de tipos que YAPL debe de cumplir:

1. **Un programa en YAPL es una secuencia de 1 o más definiciones de clases, con atributos y métodos.**
 - a. **Los atributos dentro de una clase deben ser declarados antes de su uso.**
 - b. **Un método dentro de una clase puede ser llamado de forma recursiva.**

Ejemplo 1: Definición de una clase simple `class ClasePrueba {Int a; Int b; Int suma() {return a + b;}}` - Un programa en YAPL podría consistir en esta única clase.

Ejemplo 2: Definición de múltiples clases `class Clase1 {Int a;}; class Clase2 {Int b;};` - Un programa en YAPL también puede consistir en múltiples clases.

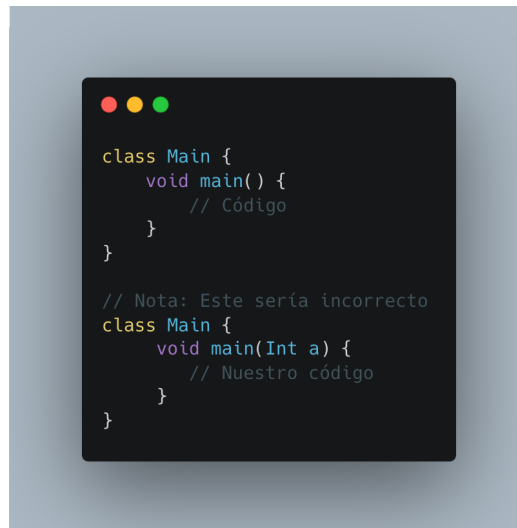
A screenshot of a code editor with a dark background and light-colored text. The code defines three classes in YAPL: 'ClasePrueba' with attributes 'a' and 'b', and a method 'suma' that returns 'a + b'; 'Clase1' with attribute 'a'; and 'Clase2' with attribute 'b'. The code is as follows:

```
class ClasePrueba {  
    Int a;  
    Int b;  
  
    Int suma() {  
        return a +  
b; }  
}  
  
class Clase1 {  
    Int a;  
}  
  
class Clase2 {  
    Int b;  
}
```

2. **Todo programa en YAPL debe contener una clase llamada "Main".**
 - a. **La clase Main debe contener un método main sin parámetros formales.**
 - b. **La clase Main no puede heredar de ninguna otra clase.**

Ejemplo 1: `class Main { void main() { /* código */ }}` - Esta es una definición correcta de la clase Main.

Ejemplo 2: `class Main { void main(Int a) { /* código */ }}` - Esto sería incorrecto, ya que el método main no debe tener parámetros.

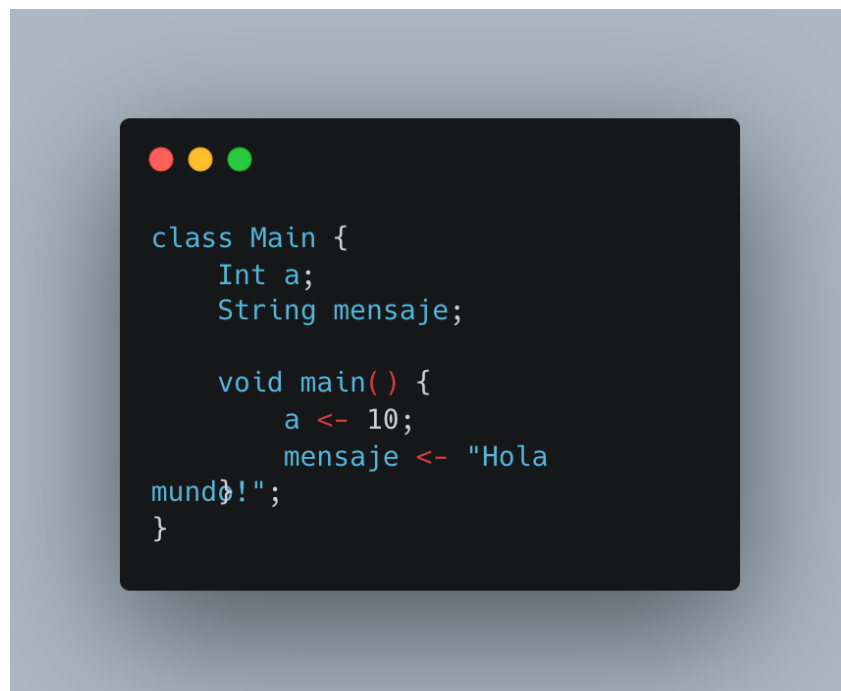


3. Los tipos básicos del lenguaje son Int, String y Bool.

- a. Int: representa números enteros
- b. String: Representa cadenas de caracteres
- c. Bool: Representa valores booleanos: True and False.

Ejemplo 1: `Int a <- 10;` - Aquí se está declarando una variable a de tipo Int con valor 10.

Ejemplo 2: `String mensaje <- "Hola mundo!";` - Aquí se está declarando una variable mensaje de tipo String con valor "Hola mundo!".

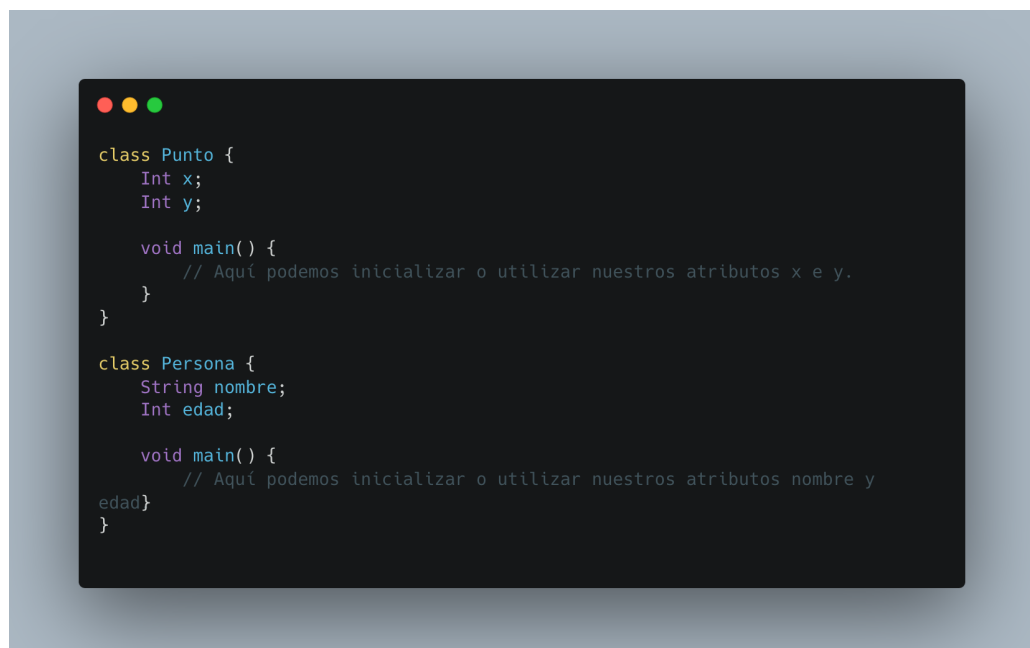


4. Los tipos básicos se pueden utilizar en la definición de clases, creando nuevos tipos derivados a partir de ellos.

- a. Las clases que definen los tipos básicos no pueden ser padres de ninguna otra clase.**

Ejemplo 1: `class Punto {Int x; Int y;}` - Aquí se está definiendo una clase Punto que utiliza los tipos básicos Int para sus atributos.

Ejemplo 2: `class Persona {String nombre; Int edad;}` - Aquí se está definiendo una clase Persona que utiliza los tipos básicos String e Int para sus atributos.



5. Existen dos ámbitos dentro de una clase: Global y Local.

- a. Los identificadores de un ámbito local ocultan la definición de identificadores en el ámbito global.**
- b. Ningún identificador puede ser definido más de una vez dentro de un mismo ámbito.**

Ejemplo 1: `class MyClass {let x: Int}` se define x en el ambito global

Ejemplo 2: `class My Class {`


`let x: Int`

`method myMethod() {`

`let x: Bool`

`}`

`}`se define 'x' como variable local y oculta a 'x' definida globalmente



```

1  class MyClass {
2      let x: Int; //ambito global
3
4      method myMethod() {
5          let x: Bool; // variable local
6          x <- true;
7          return x; // se refiere al bool
8      }
9
10     method anotherMethod() {
11         x <- 42; // se refiere al atributo
12     }
13 }

```

- 6. Si una clase B hereda de una clase A y B sobrescribe un método de A, este método debe tener la misma firma que fue declarada en A.**

Ejemplo 1: class A {method calculate(x:int, y:int)}; class B extends A {method calculate(x:int, y:int)}; - Esto sería correcto, ya que el metodo de clase B coincide con la firme de clase A

Ejemplo 2: class A {method calculate(x:int, y:int)}; class C extends A {method calculate(x:Bool, y:int)}; - Esto sería incorrecto, ya que el metodo de clase C no coincide con la firme de clase A

```
1 class A {
2     method calculate(x: Int, y: Int) {
3         return x + y;
4     }
5 }
6
7 class B extends A {
8     //correcto ya que coincide con la firma de la clase A
9     method calculate(x: Int, y: Int) {
10         return x - y;
11     }
12 }
13
14 class C extends A {
15     //incorrecto ya que no coincide con la firma de la clase A
16     method calculate(x: Bool, y: Int) {
17         return x * y;
18     }
19 }
```

7. No es posible la herencia múltiple de clases y herencia recursiva.

Ejemplo 1: class A { }; class B extends A { }; class C extends B { }; - Esto es correcto, ya que no hay herencia múltiple ni recursiva.

Ejemplo 2: class A extends B { }; class B extends A { }; - Esto sería incorrecto, ya que hay herencia recursiva.

```
class A { }
class B extends A { }
class C extends B { }

// Nota: Esto sería incorrecto
class A extends B { }
class B extends A { }
```

8. Valores por defecto

- a. Los objetos creados a partir de la clase Int tienen valor por defecto 0.

- b. Los objetos creados a partir de la clase String tienen valor por defecto "" (cadena vacía).
- c. Los objetos creados a partir de la clase Bool tienen valor por defecto false.

Ejemplo 1: var numero: Int; numero será igual a 0

Ejemplo 2: var cadena: String; cadena será igual a "".

Ejemplo 3: var siEs: Bool; siEs será igual a false;



```
1 class MyClass {  
2     var numero: Int;  
3     var cadena: String;  
4     var siEs: Bool;  
5  
6     method displayValues() {  
7         writeLine("numero: " + numero);  
8         writeLine("cadena: " + cadena);  
9         writeLine("siEs: " + siEs);  
10    }  
11 }
```

9. Es posible el casteo implícito de Bool a Int. (False es 0, True es 1).
- a. También es posible el casteo de Int a Bool (0 es False, cualquier valor positivo es True).

Ejemplo 1: Int a <- True; - Esto es correcto, a tomará el valor 1 después de esta operación.

Ejemplo 2: Int b <- False; - Esto es correcto, b tomará el valor 0 después de esta operación.

```

class Main {
  Int a;
  Int b;

  void main() {
    a <- True; // Esto es correcto. 'a' tomará el valor 1
    b <- False; // Esto es correcto. 'b' tomará el valor 0
  }
}

```

10. No es posible el casteo explícito.

Ejemplo 1: a: Int; a <- 43, correcto, se asigna otro valor del mismo tipo

Ejemplo 2: a: Int; b: Bool; B <- (Bool)a; incorrecto, no se puede una transformación forzada

```

1  class Main {
2    a : Int;
3    b : Bool;
4
5    void main() {
6      a <- 43; //correcto ya que es el mismo tipo
7      b <- (Bool)a; //incorrecto, un int no puede ser un bool
8    }
9  }

```

11. El tipo estático <expr> debe coincidir con el tipo declarado para el <id>, o ser de un tipo heredado a partir del tipo de <id>.

Ejemplo 1: Int a <- 10; - Esto es correcto ya que el tipo estático de 10 es Int, que coincide con el tipo declarado de a.

Ejemplo 2: String mensaje <- a; - Esto sería incorrecto (asumiendo que a sigue siendo de tipo Int), ya que el tipo estático de a no coincide con el tipo declarado de mensaje.


```

class Main {
  Int a;
  String mensaje;

  void main() {
    a <- 10; // Esto es correcto.

    mensaje <- a; // Esto sería incorrecto. El tipo estático de a es
  Int }
}

```

12. El tipo de la expresión de retorno del método debe coincidir con el tipo de retorno declarado con el método.

Ejemplo 1: `Int suma (Int a, Int b) {return a + b;}` - Esto es correcto ya que la suma de a y b es de tipo Int, que coincide con el tipo de retorno del método.

Ejemplo 2: `String getNombre() {return nombre;}` - Esto sería incorrecto si nombre no es de tipo String, ya que el tipo de retorno del método es String.

```

class Main {
  Int a;
  Int b;
  String nombre;

  Int suma(Int a, Int b) {
    return a + b; // Esto es correcto. La suma de 'a' y 'b' es de tipo Int.
  }

  String getNombre() {
    return nombre; // Esto sería incorrecto si 'nombre' no es de tipo
String.
  }

  void main() {
    a <- 5;
    b <- 10;
    nombre <- "YAPL";

    Int resultadoSuma <- suma(a, b);
    String miNombre <- getNombre();
  }
}

```

13. El tipo de dato estático de la <expr> utilizada en una estructura de control if o while debe ser de tipo Bool.

Ejemplo 1: if (True) { /* bloque de código */ } - Esto es correcto, ya que la expresión dentro del if es de tipo Bool.

Ejemplo 2: if (10) { /* bloque de código */ } - Esto sería incorrecto, ya que la expresión dentro del if no es de tipo Bool.



```
class Main {
    Bool a;

    void main() {
        a <- True;


        if (a) {
            // Bloque de código
        }

        // Nota: Esto sería
        incorrecto Int b <- 10;
        if (b) {
            // Bloque de código
        }
    }
}
```

14. Los operadores de comparación se aplican a Objetos que del mismo tipo de datos estático o que sean Objetos de clases heredadas de la misma clase.

Ejemplo 1: Int a <- 10; Int b <- 20; Bool result <- a < b; - Esto es correcto, ya que a y b son del mismo tipo.

Ejemplo 2: Int a <- 10; String b <- "20"; Bool result <- a < b; - Esto sería incorrecto, ya que a y b no son del mismo tipo.



```

class Main {
    Int a;
    Int b;
    Bool result;

    void main() {
        a <- 10;
        b <- 20;

        // Esto es correcto, ya que a y b son del mismo
tipo.    result <- a < b;
        // Nota: Esto seria incorrecto
        String c <- "20";
        result <- a < c;
    }
}

```

Definiendo el sistema de tipos que YAPL debe de cumplir:

1. Tipos básicos: Int, String y Bool.
2. Tipos derivados: Se pueden crear nuevos tipos derivados a partir de los tipos básicos.
3. Herencia: Una clase puede heredar de otra clase, y debe cumplir con las reglas de herencia mencionadas en las reglas de tipos.
4. Casteo implícito: Es posible el casteo implícito de Bool a Int y de Int a Bool, según las reglas de tipos.
5. Casteo explícito: No está permitido el casteo explícito.
6. Asignaciones: Las asignaciones deben cumplir con las reglas de tipos mencionadas anteriormente.
7. Ámbitos: Existen ámbitos Global y Local dentro de una clase.
8. Métodos: Los argumentos de los métodos de tipos básicos se pasan por valor, mientras que los argumentos de los métodos de tipos derivados se pasan por referencia.
9. Valores de retorno: El tipo de la expresión de retorno de un método debe coincidir con el tipo de retorno declarado para el método.
10. Estructuras de control: El tipo de dato estático de la expresión utilizada en estructuras de control if o while debe ser de tipo Bool.
11. Clases especiales: Existe una clase especial IO que define funciones de entrada y salida de valores tipo Int y Bool.

Diseño de estructura de datos para la tabla de símbolos

Para la tabla de símbolos, se crearán dos clases distintas, una clase será el Objeto Símbolo y la otra clase será el Objeto TablaDeSímbolos.

1. Un símbolo será cualquier identificador que se utilice en el programa, ya sea una variable o un método. Los atributos que tendrá un símbolo serán los siguientes:
 - a. name: este será el nombre que se le dará al identificador.
 - b. data_type: este será el tipo de dato que es el identificador.
 - c. value: este será el valor inicial que se le da al identificador.
 - d. De ser necesario se pueden agregar más atributos a la clase, dependiendo de la información que se necesite utilizar y almacenar de cada dato.

La clase símbolo también contará con funciones para poder modificar un símbolo específico.

- a. update_value(self, new_value): función para poder modificar el valor actual de un símbolo.
 - b. update_data_type(self, new_data_type): función para poder modificar el tipo de dato de un símbolo.
2. Una tabla de símbolos será una colección de símbolos, que se almacenaran por su nombre, tipo y valor. La tabla almacenará todos los símbolos en una estructura de datos de tipo diccionario, la cual almacena el nombre del identificador como clave, y todo el símbolo como valor. El diccionario de símbolos será el único atributo que tiene este objeto. Los métodos que tendrá este objeto serán los siguientes:
 - a. insert(self, symbol): Para poder insertar un nuevo símbolo al diccionario. Esta función cuenta con verificación de tipos de datos, para poder asegurarnos que los tipos sean coherentes.
 - b. lookup(self, name): Para poder retraer un símbolo del diccionario con base a su nombre.
 - c. remove(self, name): Para poder eliminar un símbolo de la tabla.
 - d. update_symbol_value(self, symbol_name, new_value): función que llama a la función update_value de la clase símbolo para modificar el valor de un símbolo especificado.
 - e. update_symbol_data_type(self, symbol_name, new_data_type): función que llama a la función update_data_type de la clase símbolo para modificar el tipo de dato de un símbolo especificado.

- f. Display(self): Para poder mostrar todos los símbolos que están almacenados en el diccionario.
- g. Todas las funciones, excepto la función display, update_symbol_value y upadate_symbol_data_type, cuentan con un control de errores para poder controlar situaciones como símbolos duplicados, referencias a símbolos duplicados, etc.

Implementación

Implementación de la clase Symbol con sus atributos y métodos definidos.

```
1 class Symbol:
2     def __init__(self, name, data_type, value):
3         self.name = name
4         self.data_type = data_type
5         self.value = value
6
7     def update_value(self, new_value):
8         if type(new_value) != type(self.value):
9             raise ValueError(f"Invalid type '{self.name}': {type(new_value)}")
10
11         self.value = new_value
12
13     def update_data_type(self, new_data_type):
14         # Verificar si el nuevo tipo es un tipo válido
15         valid_data_types = ['int', 'float', 'string', 'bool'] #
16         if new_data_type not in valid_data_types:
17             raise ValueError(f"Invalid data type '{self.name}': {new_data_type}")
18
19         if new_data_type != self.data_type:
20             self.data_type = new_data_type
21
22     def __str__(self):
23         return f"{self.name}: {self.data_type} = {self.value}"
```

Implementación de la clase SymbolTable con sus atributos y métodos definidos.

```
1  from prettytable import PrettyTable
2
3  class SymbolTable:
4      def __init__(self):
5          self.symbols = {}
6
7      def insert(self, symbol):
8          if symbol.name in self.symbols:
9              raise ValueError(f"Symbol '{symbol.name}' already exists in the table.")
10
11         # Verificar tipo válido (opcional)
12         valid_data_types = ['int', 'float', 'string', 'bool'] # Lista de tipos válidos
13         if symbol.data_type not in valid_data_types:
14             raise ValueError(f"Invalid data type for symbol '{symbol.name}': {symbol.data_type}")
15
16         self.symbols[symbol.name] = symbol
17
18     def lookup(self, name):
19         symbol = self.symbols.get(name, None)
20         if symbol is None:
21             raise ValueError(f"Symbol '{name}' not found in the table.")
22         return symbol
23
24     def remove(self, name):
25         if name not in self.symbols:
26             raise ValueError(f"Symbol '{name}' not found in the table.")
27         del self.symbols[name]
28
29     def update_symbol_value(self, symbol_name, new_value):
30         symbol = self.lookup(symbol_name)
31         symbol.update_value(new_value)
32
33     def update_symbol_data_type(self, symbol_name, new_data_type):
34         symbol = self.lookup(symbol_name)
35         symbol.update_data_type(new_data_type)
36
37     def display(self):
38         table = PrettyTable()
39         table.field_names = ["Name", "Data Type", "Value"]
40
41         for symbol in self.symbols.values():
42             table.add_row([symbol.name, symbol.data_type, symbol.value])
43
44         print(table)
```

Ejemplo de uso de la implementación y output.



```
1  # Ejemplo de uso
2  if __name__ == "__main__":
3      table = SymbolTable()
4
5      symbol1 = Symbol("x", "int", 42)
6      symbol2 = Symbol("y", "float", 3.14)
7
8      try:
9          table.insert(symbol1)
10         table.insert(symbol2)
11         table.insert(symbol1)
12     except ValueError as e:
13         print(f"Error: {e}")
14
15     try:
16         table.lookup("z")
17     except ValueError as e:
18         print(f"Error: {e}")
19
20     table.display()
```



```
1  Error: Symbol 'x' already exists in the table.
2  Error: Symbol 'z' not found in the table.
3  +-----+-----+-----+
4  | Name | Data Type | Value |
5  +-----+-----+-----+
6  |  x   |    int   |   42   |
7  |  y   |   float  |  3.14  |
8  +-----+-----+-----+
```