Universidad Autónoma de Yucatán

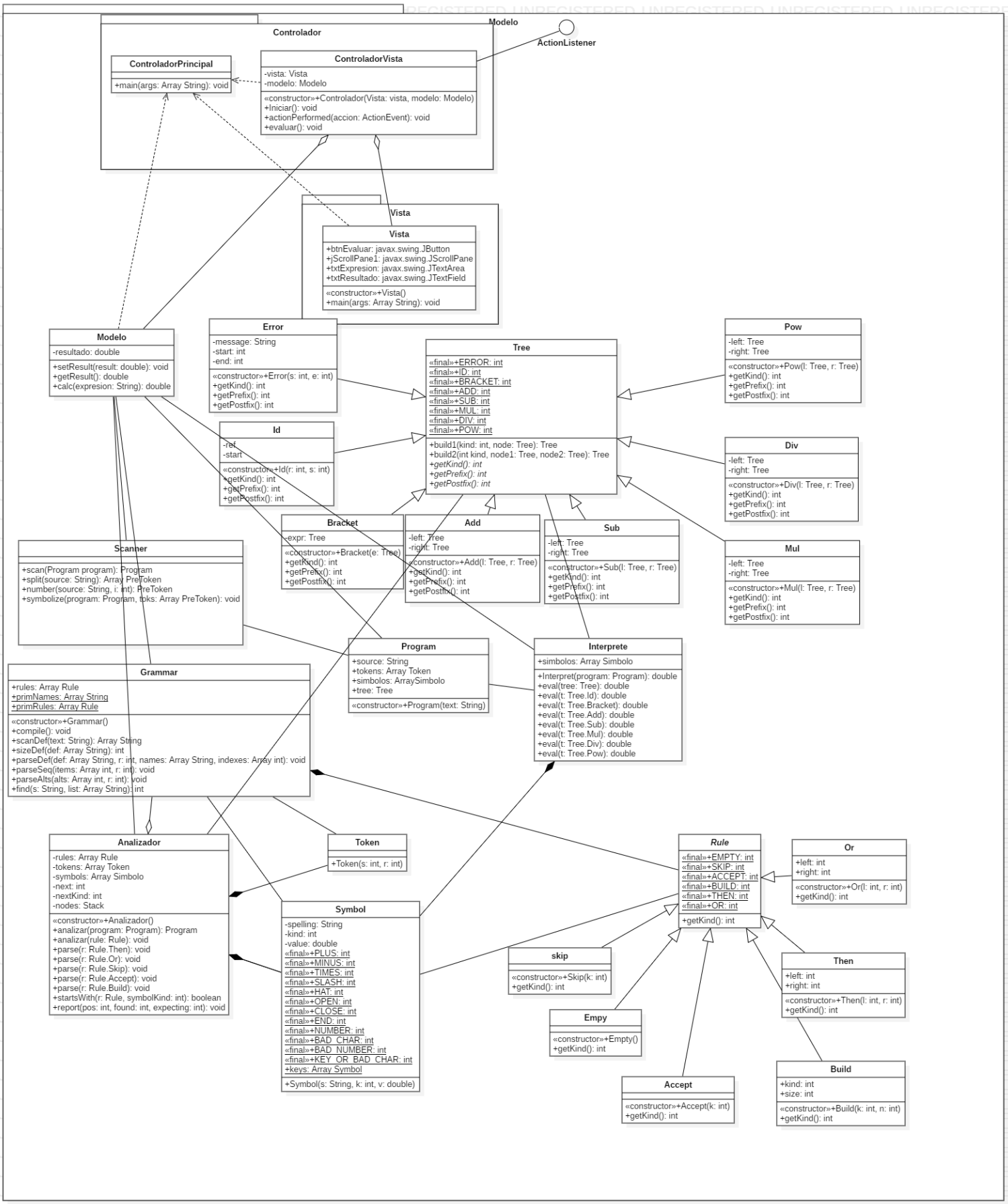
Facultad de Matemáticas

Construcción de Software

Proyecto Integrador: Calculadora numérica

Integrantes:
Luis Alberto Medina Anguas
Esteban Madrazo Parra
Armando Valdivieso Peralta
José Manuel Patrón Estrada
Angel Alexis Gonzalez Ramirez

Github: https://github.com/luisalbertomedina/calculadora


Controlador
Modelo
ActionListener
ControladorPrincipal
+main(args: Array String): void
ControladorVista
-vista: Vista
-modelo: Modelo
«constructor»+Controlador(Vista: vista, modelo: Modelo)
+Iniciar(): void
+actionPerformed(accion: ActionEvent): void
+evaluar(): void
Vista
Vista
+btnEvaluar: javax.swing.JButton
+jScrollPane1: javax.swing.JScrollPane
+txtExpresion: javax.swing.JTextArea
+txtResultado: javax.swing.JTextField
«constructor»+Vista()
+main(args: Array String): void
Modelo
-resultado: double
+setResult(result: double): void
+getResult(): double
+calc(expresion: String): double
Error
-message: String
-start: int
-end: int
«constructor»+Error(s: int, e: int)
+getKind(): int
+getPrefix(): int
+getPostfix(): int
Tree
«final»+ERROR: int
«final»+ID: int
«final»+BRACKET: int
«final»+ADD: int
«final»+SUB: int
«final»+MUL: int
«final»+DIV: int
«final»+POW: int
+build1(kind: int, node: Tree): Tree
+build2(int kind, node1: Tree, node2: Tree): Tree
+getKind(): int
+getPrefix(): int
+getPostfix(): int
Pow
-left: Tree
-right: Tree
«constructor»+Pow(l: Tree, r: Tree)
+getKind(): int
+getPrefix(): int
+getPostfix(): int
Id
-ref
-start
«constructor»+Id(r: int, s: int)
+getKind(): int
+getPrefix(): int
+getPostfix(): int
Div
-left: Tree
-right: Tree
«constructor»+Div(l: Tree, r: Tree)
+getKind(): int
+getPrefix(): int
+getPostfix(): int
Bracket
-expr: Tree
«constructor»+Bracket(e: Tree)
+getKind(): int
+getPrefix(): int
+getPostfix(): int
Add
-left: Tree
-right: Tree
«constructor»+Add(l: Tree, r: Tree)
+getKind(): int
+getPrefix(): int
+getPostfix(): int
Sub
-left: Tree
-right: Tree
«constructor»+Sub(l: Tree, r: Tree)
+getKind(): int
+getPrefix(): int
+getPostfix(): int
Mul
-left: Tree
-right: Tree
«constructor»+Mul(l: Tree, r: Tree)
+getKind(): int
+getPrefix(): int
+getPostfix(): int
Scanner
+scan(Program program): Program
+split(source: String): Array PreToken
+number(source: String, i: int): PreToken
+symbolize(program: Program, toks: Array PreToken): void
Program
+source: String
+tokens: Array Token
+simbolos: ArraySimbolo
+tree: Tree
«constructor»+Program(text: String)
Interprete
+simbolos: Array Simbolo
+Interpret(program: Program): double
+eval(tree: Tree): double
+eval(t: Tree.Id): double
+eval(t: Tree.Bracket): double
+eval(t: Tree.Add): double
+eval(t: Tree.Sub): double
+eval(t: Tree.Mul): double
+eval(t: Tree.Div): double
+eval(t: Tree.Pow): double
Grammar
+rules: Array Rule
+primNames: Array String
+primRules: Array Rule
«constructor»+Grammar()
+compile(): void
+scanDef(text: String): Array String
+sizeDef(def: Array String): int
+parseDef(def: Array String, r: int, names: Array String, indexes: Array int): void
+parseSeq(items: Array int, r: int): void
+parseAlts(alts: Array int, r: int): void
+find(s: String, list: Array String): int
Analizador
-rules: Array Rule
-tokens: Array Token
-symbols: Array Simbolo
-next: int
-nextKind: int
-nodes: Stack
«constructor»+Analizador()
+analizar(program: Program): Program
+analizar(rule: Rule): void
+parse(r: Rule.Then): void
+parse(r: Rule.Or): void
+parse(r: Rule.Skip): void
+parse(r: Rule.Accept): void
+parse(r: Rule.Build): void
+startsWith(r: Rule, symbolKind: int): boolean
+report(pos: int, found: int, expecting: int): void
Token
+Token(s: int, r: int)
Symbol
-spelling: String
-kind: int
-value: double
«final»+PLUS: int
«final»+MINUS: int
«final»+TIMES: int
«final»+SLASH: int
«final»+HAT: int
«final»+OPEN: int
«final»+CLOSE: int
«final»+END: int
«final»+NUMBER: int
«final»+BAD_CHAR: int
«final»+BAD_NUMBER: int
«final»+KEY_OR_BAD_CHAR: int
+keys: Array Symbol
+Symbol(s: String, k: int, v: double)
Rule
«final»+EMPTY: int
«final»+SKIP: int
«final»+ACCEPT: int
«final»+BUILD: int
«final»+THEN: int
«final»+OR: int
+getKind(): int
Or
+left: int
+right: int
«constructor»+Or(l: int, r: int)
+getKind(): int
skip
«constructor»+Skip(k: int)
+getKind(): int
Then
+left: int
+right: int
«constructor»+Then(l: int, r: int)
+getKind(): int
Empy
«constructor»+Empty()
+getKind(): int
Accept
«constructor»+Accept(k: int)
+getKind(): int
Build
+kind: int
+size: int
«constructor»+Build(k: int, n: int)
+getKind(): int

Checklist deConstrucción:

- **Técnicas de Construcción de Variables y tipos de datos fundamentales.**
  - Inicializar cada variable donde sea declarada:

```
String expresion = vista.txtExpresion.getText();
```

  - Inicializar cada Variable de acuerdo a dónde se utiliza por primera vez:

```
private void evaluar() throws Exception {
    String expresion = vista.txtExpresion.getText();
    vista.txtResultado.setText(String.valueOf(modelo.calc(expresion)));
}
```

  - Inicializa las constantes nominales una vez, inicializa las variables con el código ejecutable:

```
private static String[] primNames = {
    "end", "plus", "minus", "times", "slash", "hat", "open", "close",
    "number", "add", "sub", "mul", "div", "pow", "bracket", "nothing"
};
```

  - Inicializar los miembros de las clases con el constructor:

```
public ControladorVista(vista vista, Modelo modelo) {
    this.vista = vista;
    this.modelo = modelo;
    this.vista.btnEvaluar.addActionListener(this);
}
```

  - Uso correcto de nombramiento de variables:

```
tokens = program.tokens;
symbols = program.simbolos;
next = 0;
nextKind = symbols[tokens[next].ref].tipooperador;
```

- **Técnicas de organización de sentencias.**
  - Organizar código de manera que sea obvio:

```
public double calc(String expresion) throws Exception {
    Program program = new Program(expresion);
    Scanner escaner = new Scanner();
    Analizador Analizar = new Analizador();
    Interprete inter = new Interprete();

    program = escaner.scan(program);
    program = Analizar.parse(program);
    this.resultado=inter.interpret(program);

    return this.resultado;
}
```

- Nombrar rutinas de manera que se entienda:

```
Skip(int kind) {
    symbolKind = kind;
}

@Override
int getKind() {
    return SKIP;
}
```

- **Técnicas de Construcción de estructuras de control de flujo.**
  - Uso de switch case:

```
switch (rule.getKind()) {
    case Rule.THEN:
        parse((Rule.Then) rule);
        break;
    case Rule.OR:
        parse((Rule.Or) rule);
        break;
    case Rule.EMPTY:
        break;
    case Rule.SKIP:
        parse((Rule.Skip) rule);
        break;
```

  - Uso de if anidados con else:

```
private void parse(Rule.Skip rule) throws Exception {
    if (nextKind == rule.symbolKind) {
        next++;
        if (nextKind != Simbolo.END) {
            nextKind = symbols[tokens[next].ref].tipooperador;
        }
    } else {
        report(tokens[next].start, nextKind, rule.symbolKind);
    }
}
```

- **Técnicas de Construcción de clases.**
  - Nombre adecuado para una clase:

```
public class ControladorVista implements ActionListener {

    private vista vista;
    private Modelo modelo;
```

- Bajo acoplamiento de una clase:

```java
public class Token {

    int start, ref;

    Token(int inicio, int r) {
        start = inicio;
        ref = r;
    }
}
```

- **Técnicas de Documentación de Código.**
  - Comentarios al inicio de una clase para explicar su propósito:

```java
/**
 * Un token consiste en la posición del recurso y la referencia al símbolo
 * (que contiene el string del operador Ej. "-" y el tipo E. 1). Los tipos de símbolos son
 */
```

  -
- **Estándares de Codificación.**
  - En todo el proyecto se utiliza el estándar de codificación de Java.