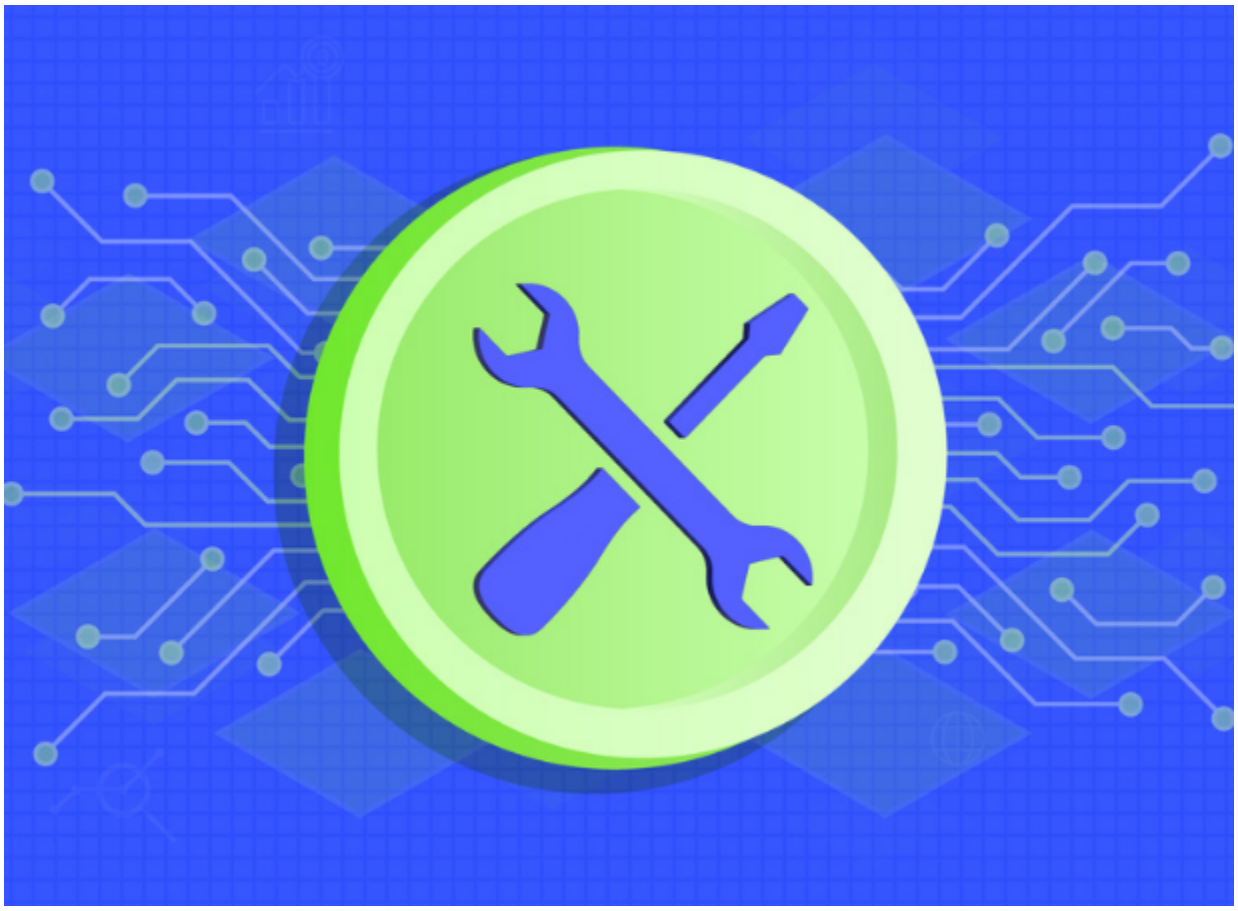
TOKEN



ÍNDICE

Objetivo de la práctica

El objetivo de la práctica es pasar expresiones matemáticas de notación infija a expresiones en notación polaca inversa. Para ello, como siempre, necesitaremos tener una serie de cosas en cuenta:

Pongamos un ejemplo con una expresión matemática:

3+5-4

Esta expresión matemática está compuesta por cinco tokens, es decir, cinco unidades matemáticas diferentes que cada una significa algo.

Primero de todo, para entender el concepto de tokens deberemos separarlos por tipos:

Los tokens númericos: Son los tokens cuyo valor es númerico, es decir, representan un número. Por ejemplo el token '5' representa el número 5. No obstante, un token numérico puede abarcar más de un número. Por ejemplo, 17 o 234.

Los tokens operadores: Son los tokens cuyo valor es un carácter que representa el tipo de operación que debemos hacer. Por ejemplo, el token '+' representa la operación de sumar. Los tokens operadores son '+, -, *, /, ^'.

Los tokens paréntesis: Son los tokens que representan los caracteres que se utilizan para poner paréntesis '(' y ')'.

TOKENS

Una vez que sabemos lo que son los tokens y para que sirve cada uno deberemos utilizar la clase tokens. Debemos hacer entender al programa que es un token y de que tipo es cada uno.

Para ello, yo he empleado la siguiente función:

```
// A partir d'un String, torna una llista de tokens
   public static Token[] getTokens(String expr) {
      List<Token> tokens = new ArrayList<>();
      String tok = "";

      String temp = "";
      for (int i = 0; i < expr.length(); i++) {
         char c = expr.charAt(i);
         if ((int) c > 47 && (int) c < 58){
            System.out.println("Es un número");
            temp += c;
            //int token = Integer.parseInt(tok);
            //return tokNumber(token[]);
         } else {
            // Hemos encontrado algo que no es un numero
            // Miramos si dentro de la variable temp hay caracteres
            // Si hay caracteres, entonces tengo que generar un nuevo token de tipo
NUMBER
            // y guardar su valor de acuerdo con estos caracteres
            if (temp.length() > 0) {
               tokens.add(Token.tokNumber(Integer.parseInt(temp)));
               temp = "";
            }
         }
```

```
            if ((int) c == '+' || (int) c == '-' || (int) c == '*' || (int) c == '/' ) {
                System.out.println("Es un operador");
                tokens.add(Token.tokOp(c));

            } else if (c == '(' || c == ')') {
                System.out.println("Es un parentesis");
                tokens.add(Token.tokParen(c));
            }
        }
        if (temp.length() > 0) {
            tokens.add(Token.tokNumber(Integer.parseInt(temp)));
        }
        Token[] tokenAr = new Token[tokens.size()];
        tokenAr = tokens.toArray(tokenAr);
return tokenAr;
    }
}
```

Esta función recibe una expresión en forma de String y tiene que transformarla en un array de Tokens.

Para ello, tendrá que diferenciar entre todos los números y caracteres que tokens hay. El primer fragmento interpreta si es un número o no dependiendo del valor del Token pasado a Integer en el código Ascii. Cuando determina que el token es un número lo guarda en una String temporal y comprueba si el siguiente token es un número. En caso de que lo sea, esto significará que es el siguiente elemento pertenece al mismo número, es decir, al mismo token. Irá comprobando hasta que el siguiente elemento ya no sea un número y cuando haga esto guardará lo que hay dentro de la String temporal en la primera posición del array de Tokens y vaciará la String. Recordemos en la lista lo guardaremos como Integer

Si lo siguiente que encuentra es un operador, haremos lo mismo que con los números: utilizando su valor en el código ascii lo identificaremos como operador y guardaremos como un operador y si encontramos un paréntesis igual. En estos casos los guardaremos como char

Ahora que sabemos que es cada elemento tenemos que transformarlos en Tokens. Para ello, he utilizado la siguiente función:

```
static Token tokOp(char c) {
```

```
    Token t = new Token();
    t.tk = c;
    t.ttype = Toktype.OP;
    return t;
  }
```

Una vez que hemos definido que tipo de token es cada elemento de la expresión que nos han mandado, lo siguiente es crear los getters, es decir, funciones que nos devuelvan el valor o el tk de cada token que hemos guardado, y para ello crearemos un getter de cada tipo. Por ejemplo en el caso de los operadores:

```
public char getTk() {
    return tk;
  }
```

El programa cuenta con un enum de Toktypes y un getter del tipo del token para devolver el tipo de token

```
enum Toktype {
    NUMBER, OP, PAREN
  }

  public Toktype getTtype() {
    return ttype;
  }
```

Para asegurarnos de que el token que nos pasan es el mismo que necesitamos y para pasar el test de tokens creamos el método equals, el cual confirma que el tipo y el tk coinciden en el caso de los operadores o paréntesis, y que el tipo y el valor coinciden en el caso de los numbers. Una vez que lo comprobamos, devolvemos el token. Sería algo así

```
public boolean equals(Object o) {
    Token t = (Token) o;
    if (t.ttype != this.ttype) return false;
    if (t.ttype == Toktype.NUMBER) {
```

```
        return t.value == this.value;
    }
    if (t.ttype == Toktype.OP) {
      return t.tk == this.tk;

    }
    if (t.ttype == Toktype.PAREN){
        return t.tk == this.tk;
    }
    return false;
}
```

EVALUATOR

La siguiente clase es la clase Evaluator, la cual tiene dos partes (rpn y calculate).

La función rpn sirve para realizar los cálculos. Nos pasan expresiones en notación polaca inversa y la función debe calcular el resultado y devolverlo.

Recordemos que una expresión en notación polaca inversa tiene este formato:

3 5 4 +- = 6

Para realizar un cálculo así primero de todo crearemos una pila o Stack. Esta pila sera de números o Integers. Después utilizaremos un bucle for que iterará el array de tokens que recibe la función e irá comprobando si los tokens recibidos son números. En caso de que lo sean se guardarán en la pila que hemos creado y así iremos guardando elementos hasta que encontremos un operador.

Cuando encontremos un operador, automáticamente extraeremos los dos últimos elementos que haya en la pila y realizaremos la operación correspondiente al operador recibido. Una vez tengamos el resultado lo guardaremos de nuevo en la pila y seguiremos iterando nuestro array de tokens.

Ese es el proceso a seguir hasta llegar al final del array. De esta manera nos aseguramos de que las operaciones se hagan entre los números que corresponde y no en otro orden, lo cual podría provocar problemas en el caso de las restas y las divisiones.

La función calculate sirve para pasar expresiones de notación infija a notación polaca inversa y para ello he creado dos pilas. Una de Integers y la otra de Characters. A partir de la expresión que recibimos del test comprobamos si el token recibido es un numero y de ser asi lo introducimos en la pila de numeros.

Para el caso de los operadores, tenemos que crear una funcion para saber la prioridad de estos. Este metodo compara la prioridad o precedencia del operador recibido con el anterior de la pila de operadores y lo baja en la pila en funcion de esa misma prioridad.

Teoricamente teniamos que pasarle las expresiones a calcRpn pero no me acordaba y he creado otra funcion que hace las operaciones dentro de Calculate que se llama performOperation.

Teniendo en cuenta la prioridad maxima de los parentesis y el resto de prioridades, simplemente se trata de ir separando en pilas y operando para obtener el resultado.