Pre-Processing

Group 01

Main Goal

Racket based pre-processor

Plus a couple of tokens

What it does?

- Local type inference
  - Only *new* expressions
  - No “Primary” nor literal support
    - `new Outer().new Inner1()...new InnerN().new ActualType()` ⇒ `Outer`
- Aliases
- String interpolation
- *`#include`*

What it doesn’t?

- The rest (You don’t say..)
  - Getter/Setter generation
  - Active token definition (outside of pre-processor, ofc)
  - C-like active tokens (macros)

But how?

Wait.. I think I’ve seen this somewhere

But where?..

Main Goal

Implement Generic Functions in Java

Generic Functions

Group 01

According to CLOS semantics

Generic Functions

Group 01

But how?

Is it magic?

Yes!.. Kind of.. Not really

Seriously, how?

Enough “memes” ._.

Fine… Moving along

File Tree

```
$ tree src
src
├── actions.rkt
│   ; Actions associated with the active tokens
├── preprocessextra.rkt
│   ; Registers extra active tokens
└── preprocess.rkt
    ; The “main engine” (where the magi- no, stop!)
```

preprocess.rkt

```racket
#lang racket
(require "actions.rkt")
(provide add-active-token def-active-token process-string)

(define-syntax-rule
  (def-active-token token (str) body)
  (add-active-token token (λ (str) body)))

(define active-tokens
  (make-hash (list (cons "#\"" string-interpolation)
                   (cons "alias" type-alias)
                   (cons "var" type-inference))))

(define (add-active-token token action)
  (hash-set! active-tokens token action))
```

preprocess.rkt

```racket
#lang racket
(define (process-string str)
  (define last 0)
  (define rx (regexp (string-join (hash-keys active-tokens) "|")))
  (do ([pos (regexp-match-positions rx str) (regexp-match-positions rx str last)])
    ([false? pos] str)
    (match pos [(list (cons start end))
      (define action (hash-ref active-tokens (substring str start end)))
      (define after (substring str end))
      (define result (action after))
      (set! last start)
      (or (and (equal? result after)
               (set! last (add1 last)))
          (set! str (string-replace str (substring str start) result)))])))
```

actions.rkt

```
#lang racket
(define (type-inference str)
  (match (regexp-match (pregexp "^\\s+[\\w$]+\\s*=\\s*new\\s+(.*?)\\(") str)
    [(list _ type) (string-append type str)]
    [else str]))

(define (type-alias str)
  (match (regexp-match #px"^\\s+([\\w$]+)\\s*=\\s*(.*?);" str)
    [(list all alias value)
     (regexp-replace* (pregexp (string-append "\\b" alias "\\b"))
                      (substring str (string-length all)) value)]
    [else str]))
```

actions.rkt

```
#lang racket
```

```
(define (string-interpolation str)
  (match (regexp-match-positions #rx"[^\\]\"" str)
    [(list (cons _ end))
     (string-append "\"" (regexp-replace* #rx"#{(.*?)}" str
                                          "\" + (\\1) + \"" 0 end))]
    [else (string-append "\"" str)]))
```

actions.rkt

```racket
#lang racket
(define (include-macro str)
 (match (regexp-match #px"^\\s+\"(.*?[^\\\\])\"" str)
   [(list all file)
    (with-handlers ([exn:fail:filesystem?
                     (λ (e) (error '\#include "Could not include ~s~%  ~a"
                                   file (exn-message e)))])
      (regexp-replace all str (file->string file #:mode 'text)))]
   [else (error '\#include "Malformed statement")]))
```

preprocessextra.rkt

```
#lang racket
(require "actions.rkt" "preprocess.rkt")
(add-active-token "#include" include-macro)
```

That’s it!

Any questions?