Part I
Typed Classes

```
#' -> ' typed-parse.rkt
   -> ClassT typed-class.rkt
      ClassT -> ClassI typecheck
      ClassI -> ClassC
         inherit.rkt
               ClassC -> '
          unparsed.rkt
```
Part 2
Typed Classes with Source Locations

```
#'
  -> ' typed-parse.rkt
    '
      ClassT typed-class.rkt

ClassT
  -> ClassI
    typecheck

ClassI
  -> ClassC

ClassC
  -> '
```

inherit.rkt
unparse.rkt
Typed Classes with Source Locations

```
#'
→ ClassT
→ ClassI
typecheck

ClassT
→ ClassI

ClassI
→ ClassC

ClassC
→ #'
```

```
typed-parse.rkt

typed-class.rkt

inherit.rkt

unparse.rkt
```
Part 3
#lang typed-class

Right now, we have to write

```racket
#lang s-exp "typed-class-lang.rkt"
....
```

We’d prefer to write

```racket
#lang typed-class
....
```
Languages

A name like `typed-class` is used as a `collection` name

... and the collection’s "main.rkt" module is used

In the simple case, a `package` implements a collection

In the simplest case, a `directory` implements a package
Languages

#lang module-path

- Find module-path

  ... adding /main if no /

- Look for a reader submodule

  ... to control #lang module-path expansion to

  (module name initial-import ....)
Part 4
Right now, we write

```rkt
#lang typed-class

....
{class posn3D extends posn
  {[z : num]}
  {mdist : num -> num
   {+ {get this z}
    {super mdist arg}}}}}
....
```

Conceivably, someone might prefer to write

```rkt
#lang typed-class/infix
...
class posn3D extends posn {
  num z;
  num mdist(num arg) {
    return this.z + super.mdist(arg);
  }
}
...
```
Parsing Characters

• Is `posn3D` an identifier? Is `3Dposn` an identifier?

• Is

\[ 1 + 2 \times 3 \]

the same as

\[ (1 + 2) \times 3 \]

or

\[ 1 + (2 \times 3) \]
Parsing Characters

• Is `posn3D` an identifier? Is `3Dposn` an identifier?

• Is

\[ \text{this.m(0)} + \text{this.n(1)} \]

the same as

\[ (\text{this.m(0)}) + (\text{this.n(1)}) \]

or

\[ (\text{this.m(0)} + \text{this}).n(1) \]

• Is `class` a reserved word?

• Is `1+2` the same as `1 + 2`?
Lexing and Parsing

Characters:

```
class <space> posn 3D ... { ... }
```

Tokens:

```
CLASS WHITESPACE posn3D ... OPENB ... 
```

```
lexer.rkt
```

```
characters
→ tokens
```

```
parser.rkt
```

```
tokens
→ abstract syntax
```
Lexing and Parsing

Characters:

```
class <space> posn 3D ... { ...
```

Tokens:

```
CLASS WHITESPACE posn3D ... OPENB ...
```

```
lexer.rkt  
```

```
characters  → tokens
```

```
parser.rkt  
```

```
tokens  → '#'
```

31
Lexing and Parsing

Characters:

```
class <space> posn 3D ...
```

Tokens:

```
CLASS WHITESPACE posn3D ... OPENB ...
```

```
lexer.rkt
```

```
characters → tokens
```

```
tokens → #'
```

```
parser.rkt
```

```
#'
→ #'(module ....)
```

```
reader.rkt
```
Part 5
Languages

```plaintext
#lang module-path
```

- Find `module-path`

- Look for a `reader` submodule
  
  ... which parses characters into `(module ....)` by providing a `read-syntax` function
Part 6
Syntax Coloring

#lang typed-class/infix0

....

(new posn3D(5, 3, 1)).addDist(new posn(2, 7));
Syntax Coloring

```
#lang typed-class/infix
....
(new posn3D(5, 3, 1)).addDist(new posn(2, 7));
```
Languages

#lang module-path

• Find module-path

• Look for a reader submodule
  ... which provides a get-info function to configure details such as syntax coloring