Typechecking Cool and PA3

Agenda

- About PA3 & Starter Code
- Typechecking the AST (simplified)
- Typechecking the AST (with rules in detail)

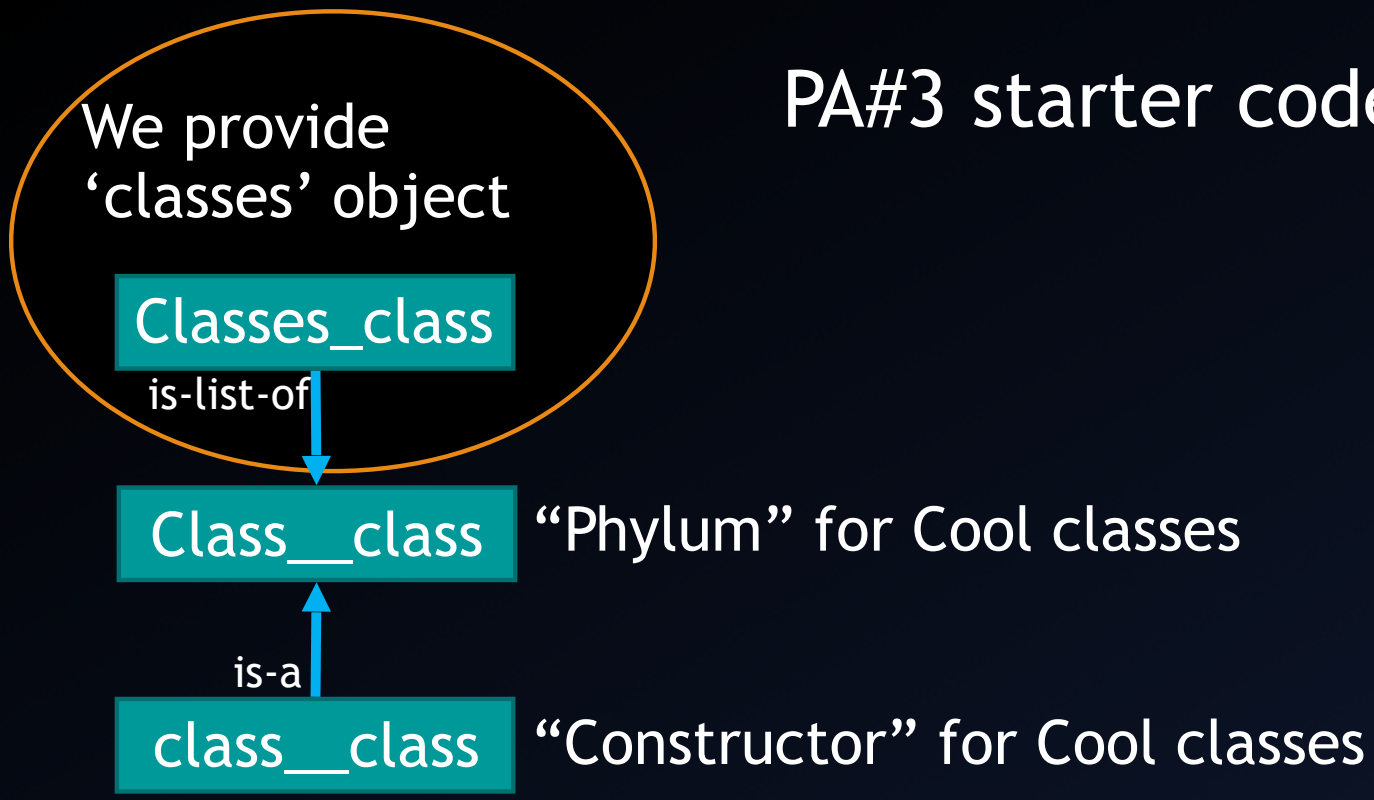
PA#3

- It’s big. More code than PA1 and PA2.
- Some tricky concepts.
- Many corner cases.
- If you haven’t started, start ASAP.

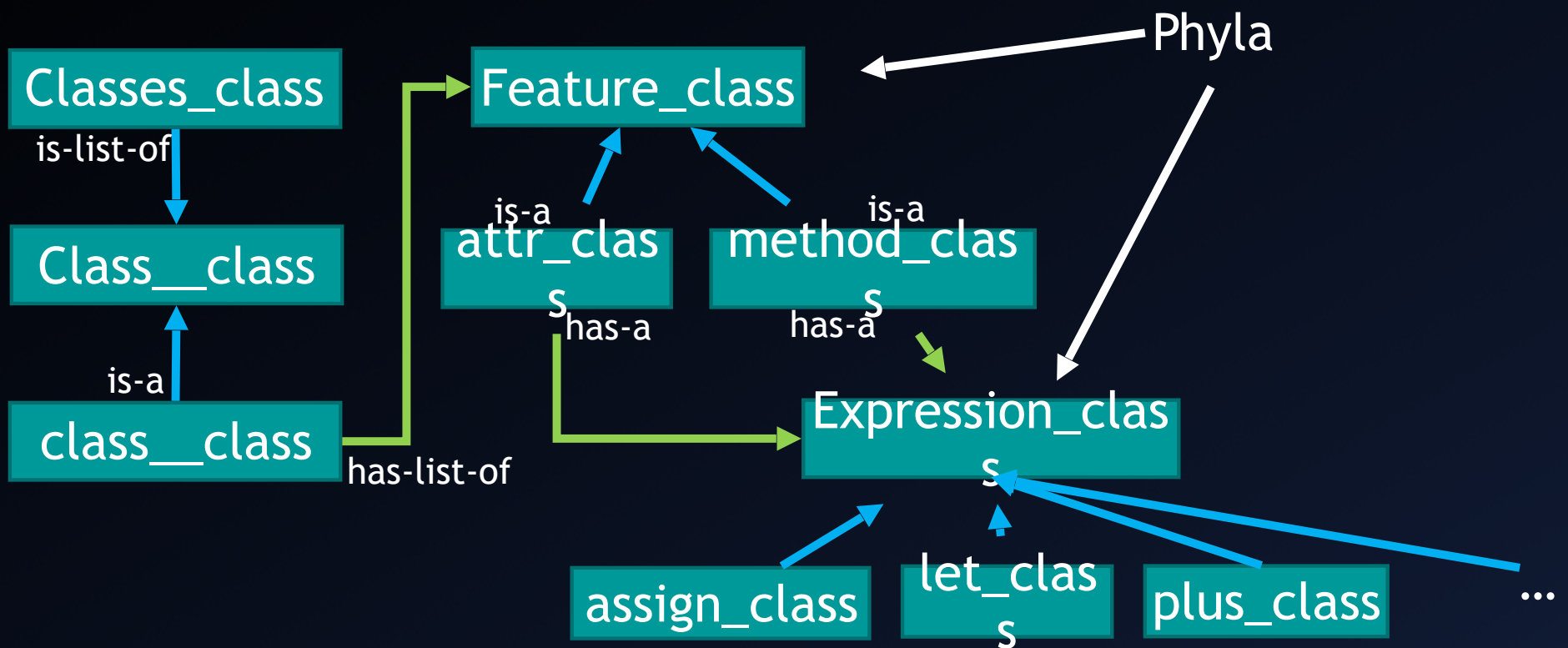
PA#3 Observations

- You have a lot of flexibility
  - Choices of data structures
  - Error messages
  - Code organization
  - Algorithmic choices (how many passes, etc.)
  - ……
- The starter code is minimal
  - You'll need to write some boilerplate
  - You should (must?) change starter code

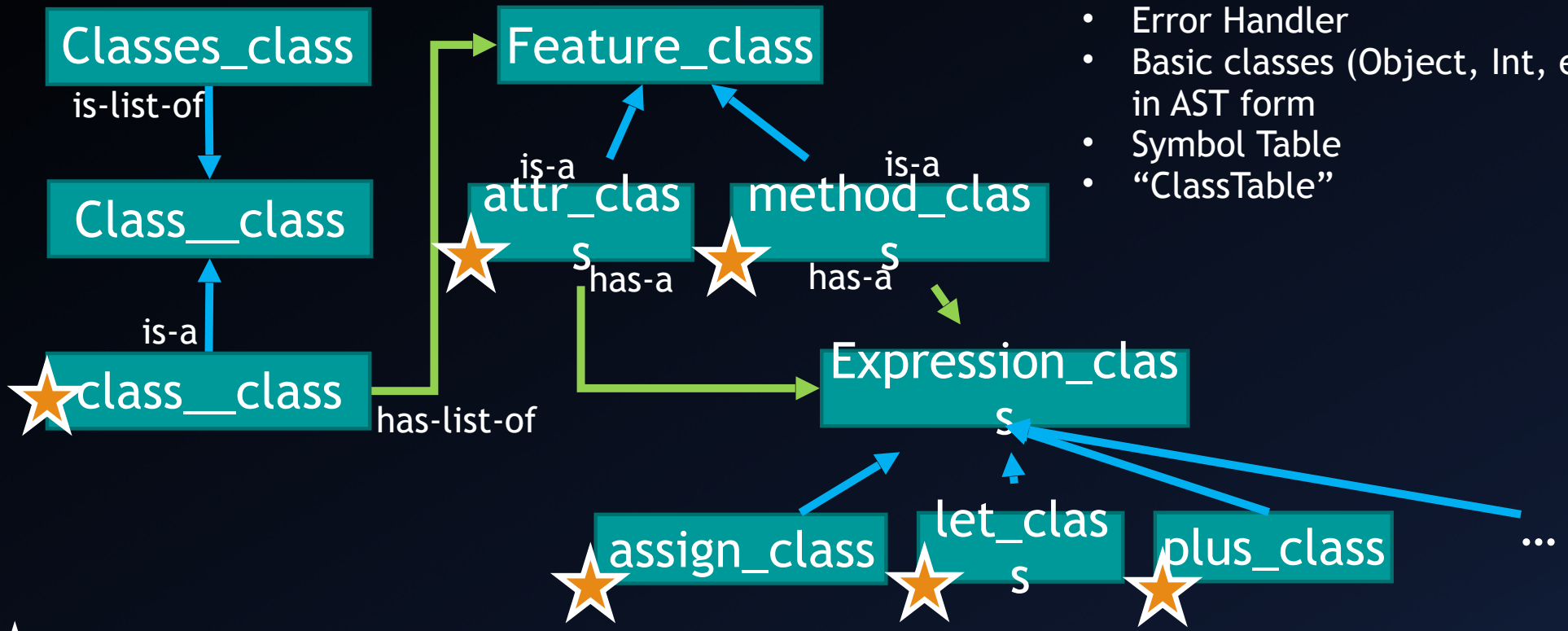
PA#3 starter code



PA#3 starter code



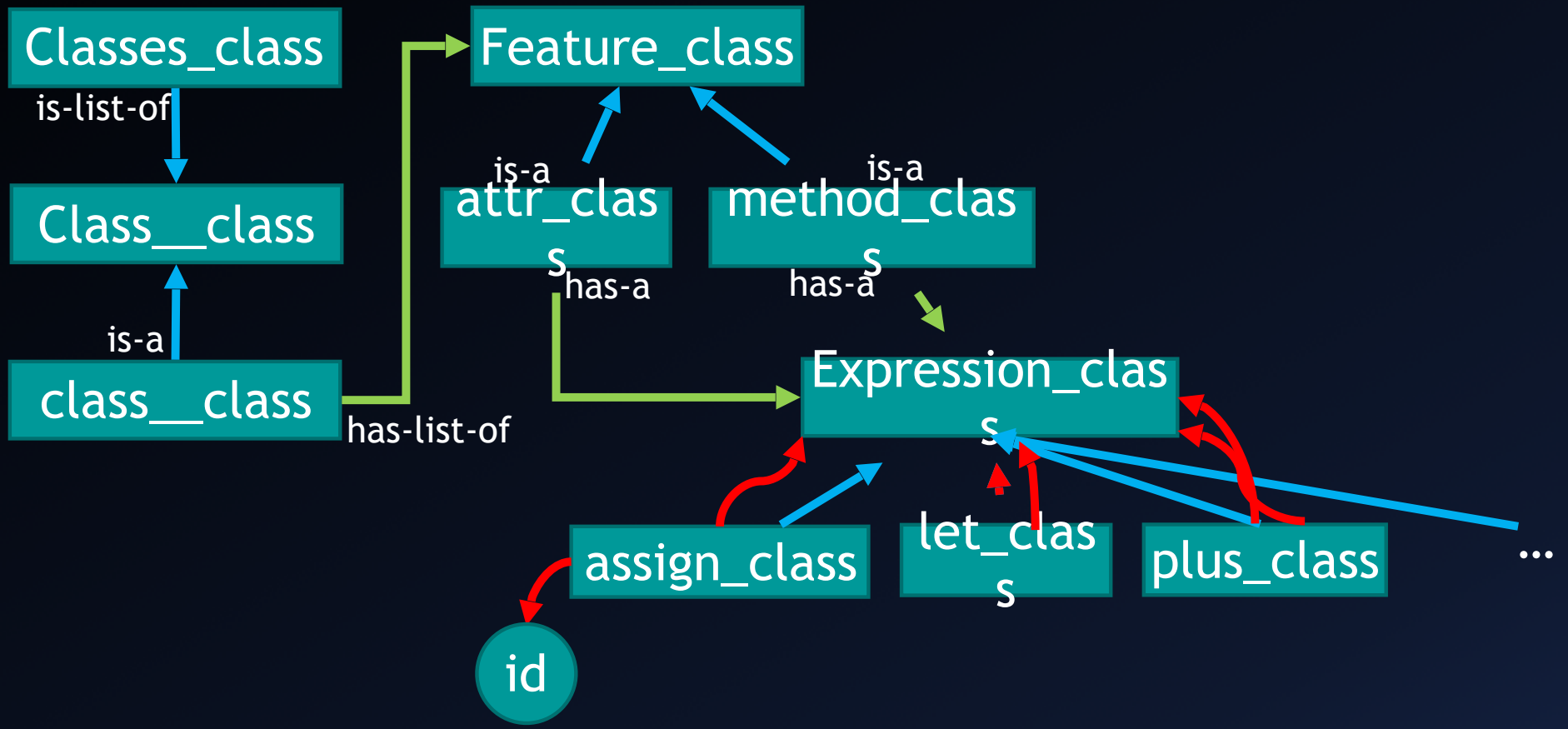
PA#3 starter code



- Error Handler
- Basic classes (Object, Int, etc.) in AST form
- Symbol Table
- “ClassTable”

constructor: has members you need to write accessors for

PA#3 starter code



PA#3 Tips

- Don’t write phases independently (to combine later)
  - The first phase(s) are much simpler than the typechecking phase.
  - Typechecking has many dependencies on first phase.
- Understand the type checking rules!

PA#3 Tips

- Before submitting, read chap 1-13 of the cool manual again.
- Look for corner cases mentioned in manual.

- Testing:
  - (common case) make sure your implementation does what you think it does!
  - (corner case) think of examples that will break your code

Example

class Foo {

bar(id_1 : $Type_1$, id_2 : $Type_2$) : $Type_3$ { …… }

};

Other than typechecking, what must we verify about bar?

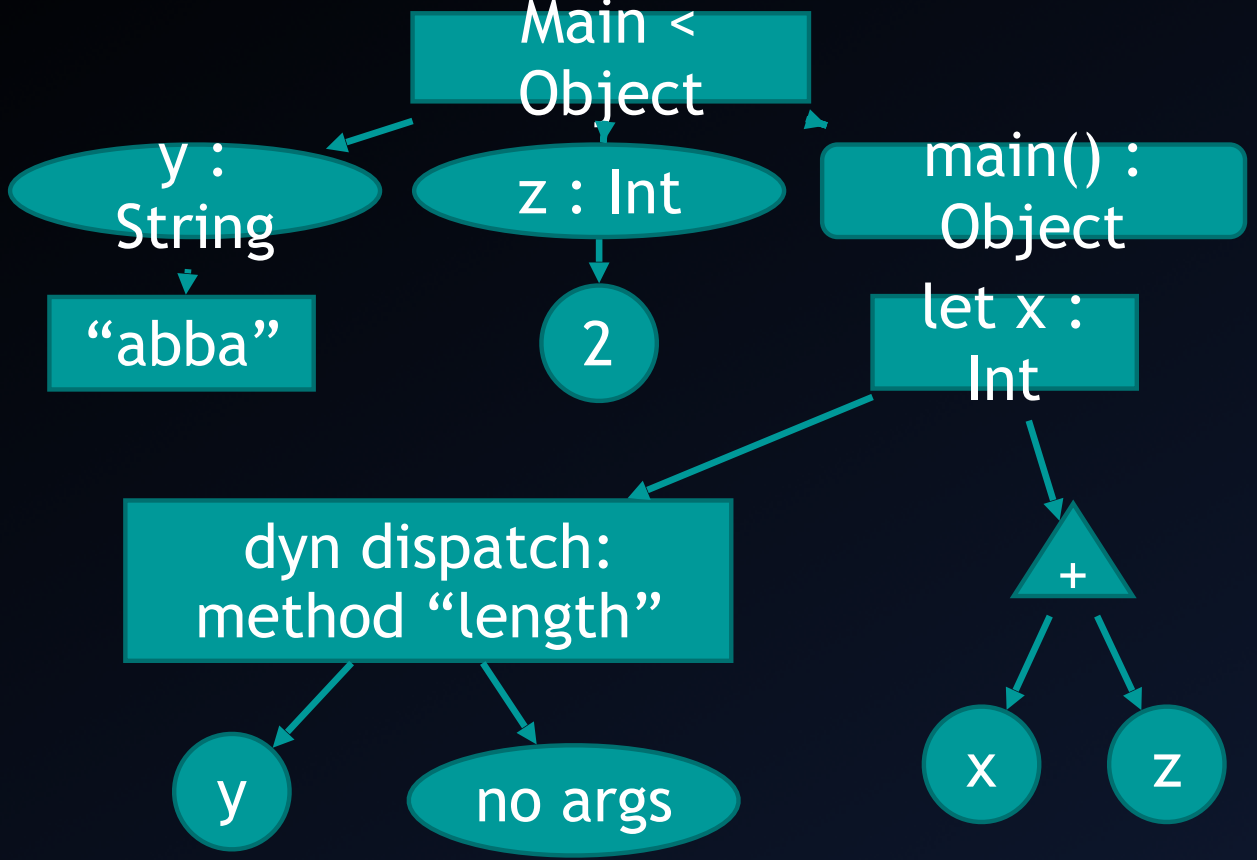
Questions on PA3 or the starter code?

Typechecking Examples

Running Example

```
class Main {
        y : String <- "abba";
        z : Int <- 2;
        main() : Object {
                let x : Int <- y.length() in x+z
        };
};
```

Running Example




```
class Main {
    y : String <- "abba";
    z : Int <- 2;
    main() : Object {
        let x : Int <-
y.length() in
                x+z
    };
};
```

Before Typechecking

- Compute “Method Environment” (M)
- Any data structure you like!
- Need to be able to lookup methods by type and name

Main.main() : Object	String.concat(String) : String
Object.cool_abort() : Object	String.length() : Int
Object.type_name() : String	String.substr(Int, Int) : String
Object.copy() : SELF_TYPE	(and all other functions…)

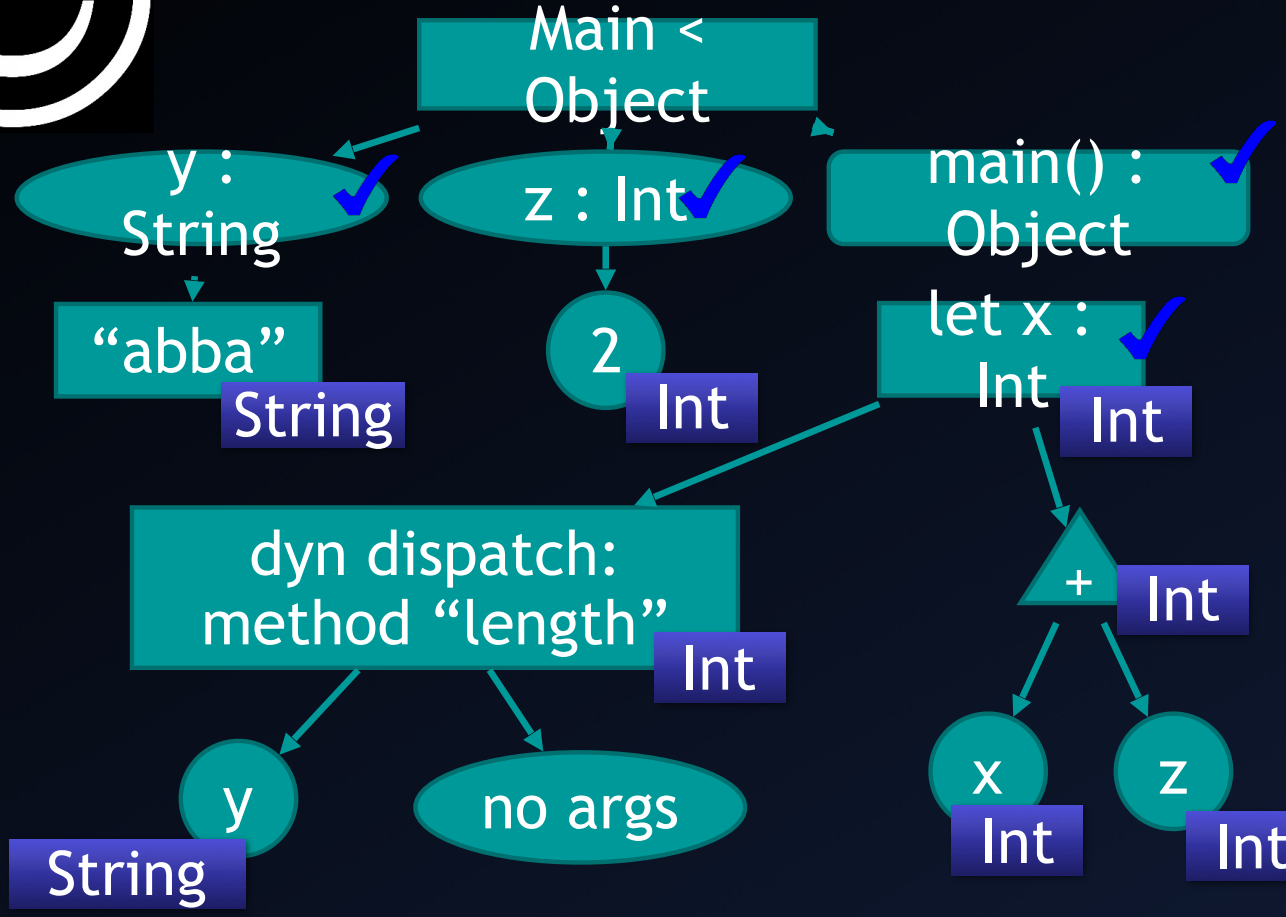
Before Typechecking

- Compute initial "Object Environment" for each class C. (O_C)
- Good idea to use symbol table (but not required!)
- Lists the attributes of the class
- Example for our Main class

O_{MAIN}

Symbol	Type
y	String
z	Int

Typechecking Example



Id	Type
y	String
z	Int
self	$SELF\_TYPE_{Main}$

Id	Type
x	Int

but what about all those rules?

Rule for Typechecking an Attribute

$$\frac{O_C(x) = T_0 \qquad O_C[SELF\_TYPE_C/self], M, C \vdash e_1 : T_1 \qquad T_1 \leq T_0}{O_C, M, C \vdash x : T_0 \leftarrow e_1}$$

- What are O, M and C?

Rule for Attributes

- O is the object environment.

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{Main}$

- M is the method environment.

Main.main() : Object	String.concat(String) : String
Object.cool_abort() : Object	String.length() : Int
Object.type_name() : String	String.substr(Int, Int) : String
Object.copy() : SELF_TYPE	(and all other functions…)

- C is the

$$\frac{O, M, C \vdash e_1 : T_1 \quad T_1 \leq T_0 \quad O[T_0/x], M, C \vdash e_2 : T_2}{O, M, C \vdash \text{let } x : T_0 \leftarrow e_1 \text{ in } e_2 : T_2}$$

To typecheck an expression, you need to know O, M and C.

- The method environment (M) is constant.
- The object environment (O) starts with O_C when typechecking C.
  - As you traverse top-to-bottom scopes are added.
- C is just the class that contains the expression.

Rule for Typechecking an Attribute

$$\frac{O_C(x) = T_0 \quad O_C[\text{SELF\_TYPE}_C/\text{self}], M, C \vdash e_1 : T_1 \quad T_1 \leq T_0}{O_C, M, C \vdash x : T_0 \leftarrow e_1}$$

- What are x, e_1, T_0, T_1?

Typechecking an Attribute

- E.g. how do we typecheck y : String <- "abba" in class Main?
- Trick: read the rule backward!

$$\frac{O_C(x) = T_0 \quad O_C[\text{SELF\_TYPE}_C/\text{self}], M, C \vdash e_1 : T_1 \quad T_1 \leq T_0}{O_C, M, C \vdash x : T_0 \leftarrow e_1}$$

- We want x -> y, T_0 -> String, e_1 -> “abba”. Don’t know T_1 yet.

y : String <- “abba” (in class Main)

- We instantiate the rule with

C=Main, x = y, T_0 = String, e_1 = “abba”

- There are three things we need to check!

$$\frac{O_C(x) = T_0 \quad O_C[\text{SELF\_TYPE}_C/\text{self}], M, C \vdash e_1 : T_1 \quad T_1 \leq T_0}{O_C, M, C \vdash x : T_0 \leftarrow e_1}$$

$$\frac{O_{Main}(y) = \text{String} \quad O_{Main}[\text{SELF\_TYPE}_{Main}/\text{self}], M, Main \vdash \text{“abba”} : T_1 \quad T_1 \leq \text{String}}{O_C, M, Main \vdash y : \text{String} \leftarrow \text{“abba”}}$$

y : String <- “abba” (in class Main)

$$\frac{O_{Main}(y) = String \quad O_{Main}[SELF\_TYPE_{Main}/self], M, Main \vdash \text{“abba”} : T_1 \quad T_1 \le String}{O_C, M, Main \vdash y : String \leftarrow \text{“abba”}}$$

- $O_{Main}(y) = String$ → Look at the table we made earlier!

Symbol	Type
y	String
z	Int

y : String <- “abba” (in class Main)

$$\frac{O_{Main}(y) = String \quad O_{Main}[SELF\_TYPE_{Main}/self], M, Main \vdash \text{“abba”} : T_1 \quad T_1 \le String}{O_{Main}, M, Main \vdash y : String \leftarrow \text{“abba”}}$$

- $O_{Main}[SELF\_TYPE_{Main}/self], M, Main \vdash \text{“abba”} : T_1$
  - We need to prove that “abba” has type T_1 in extended environment
  - We need not know what T_1 is (yet).
  - This is where the typechecker makes a recursive call.

Typechecking “abba”

- We need to prove “abba” has type T_0
- with this augmented object environment,

Symbol	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$

- Only one rule applies to a string constant:

$$\frac{x \text{ is string constant}}{O, M, C \vdash x : String}$$

- We can return “String” – no further checks required.

Now, replace T_1 by String

y : String <- “abba” (in class Main)

$$\frac{O_{Main}(y) = String \quad O_{Main}[SELF\_TYPE_{Main}/self], M, Main \vdash \text{“abba”} : T_1 \quad T_1 \leq String}{O_{Main}, M, Main \vdash y : String \leftarrow \text{“abba”}}$$

T_1 replaced by String

y : String <- “abba” (in class Main)

$$\frac{O_{Main}(y) = String \quad O_{Main}[SELF\_TYPE_{Main}/self], M, Main \vdash \text{“abba”} : String \quad String \le String}{O_{Main}, M, Main \vdash y : String \leftarrow \text{“abba”}}$$

- $O_{Main}[SELF\_TYPE_{Main}/self], M, Main \vdash \text{“abba”} : T_1$
  - We just proved this with the recursive call.
- String ≤ String is true since ≤ is reflexive.

A complete proof

$$\frac{\text{“abba” is a string constant}}{O_{Main}\ [SELF\_TYPE_{Main}/self],\ M,\ Main \vdash \text{“abba”} : String}$$

$$\frac{\dfrac{O_{Main}\ (y) = String}{O_{Main}\ [SELF\_TYPE_{Main}/self],\ M,\ Main \vdash \text{“abba”} : String} \qquad String \leq String}{O_{Main},\ M,\ Main \vdash y : String \leftarrow \text{“abba”}}$$

Last example: typechecking main()

Last example

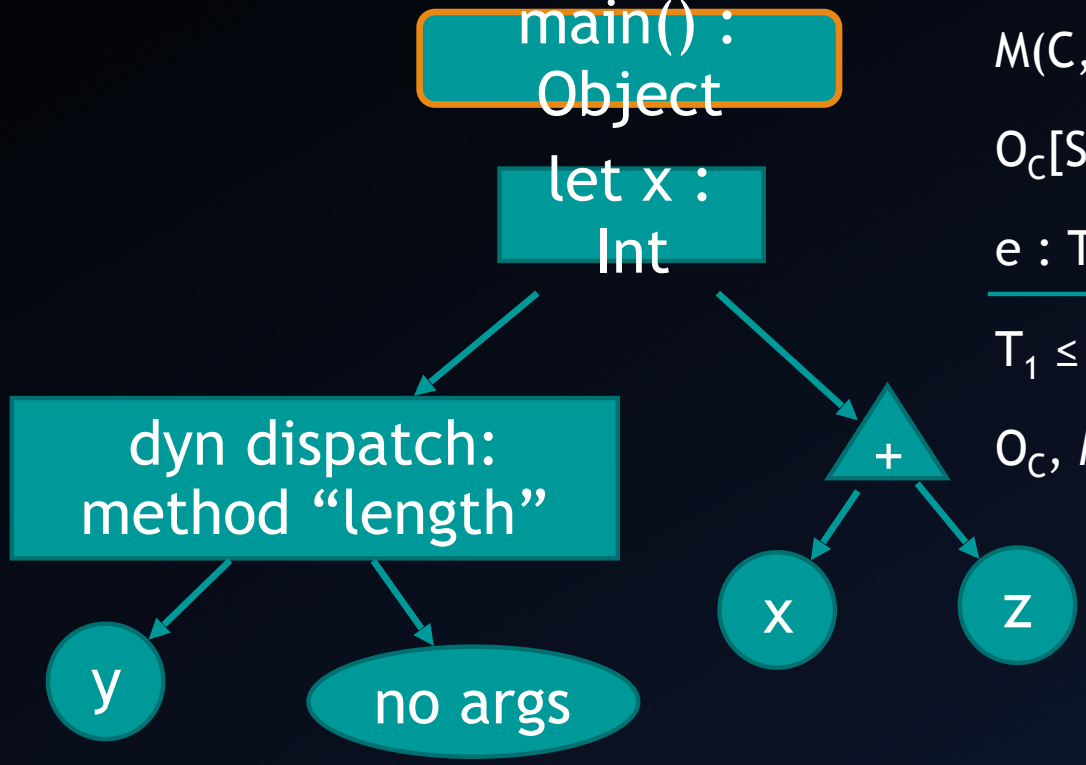
- Type check main()
- Going to simplify the rules slightly

  (will only consider some of the SELF_TYPE cases)

```
class Main {
    y : String <- "abba";
    z : Int <- 2;
    main() : Object {
        let x : Int <-
y.length() in
            x+z
    };
};
```

Typechecking Example

Id	Type
y	String
z	Int



[Method]

$M(C,f) = (T_1, T_2, \ldots, T_n, T_0)$

$O_C[SELF\_TYPE_C/self][T_1/x_1]\ldots[T_n/x_n], M, C \vdash e : T_1$

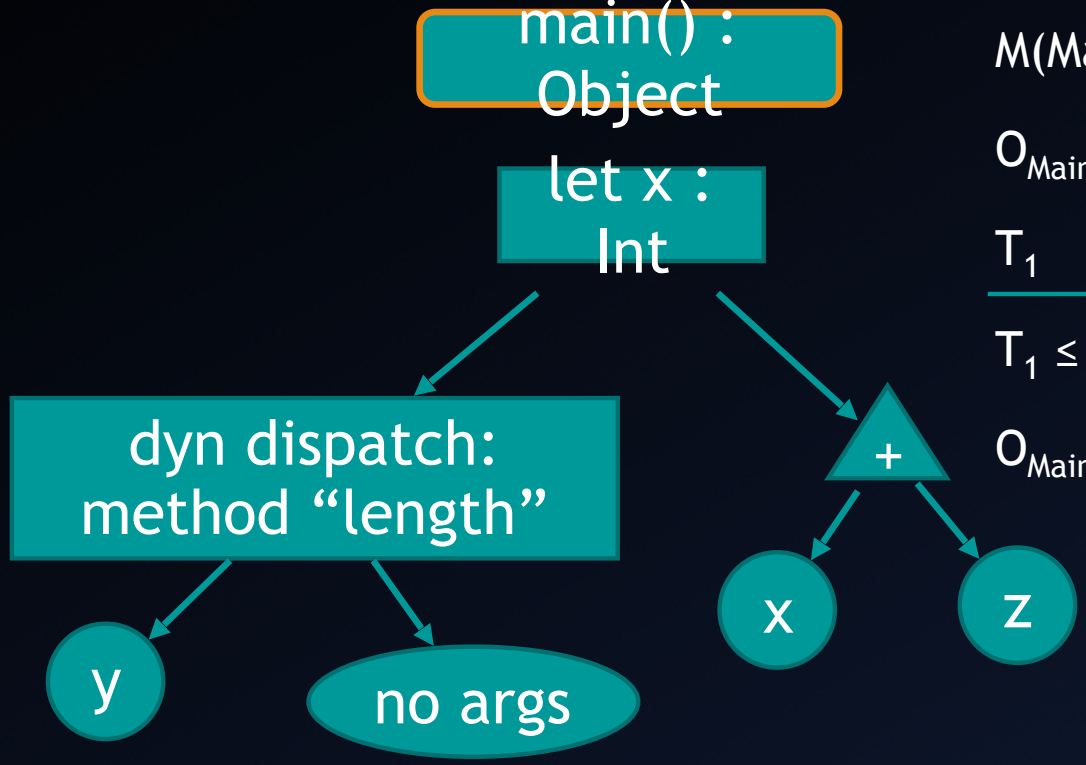
$T_1 \leq T_0$

$O_C, M, C \vdash f(x_1 : T_1, \ldots, x_n : T_n) : T_0 \{ e \}$

C -> Main
f -> main
n -> 0
e -> let x : Int ….

Typechecking Example

Id	Type
y	String
z	Int



[method]

$M(Main, main) = (T_0)$

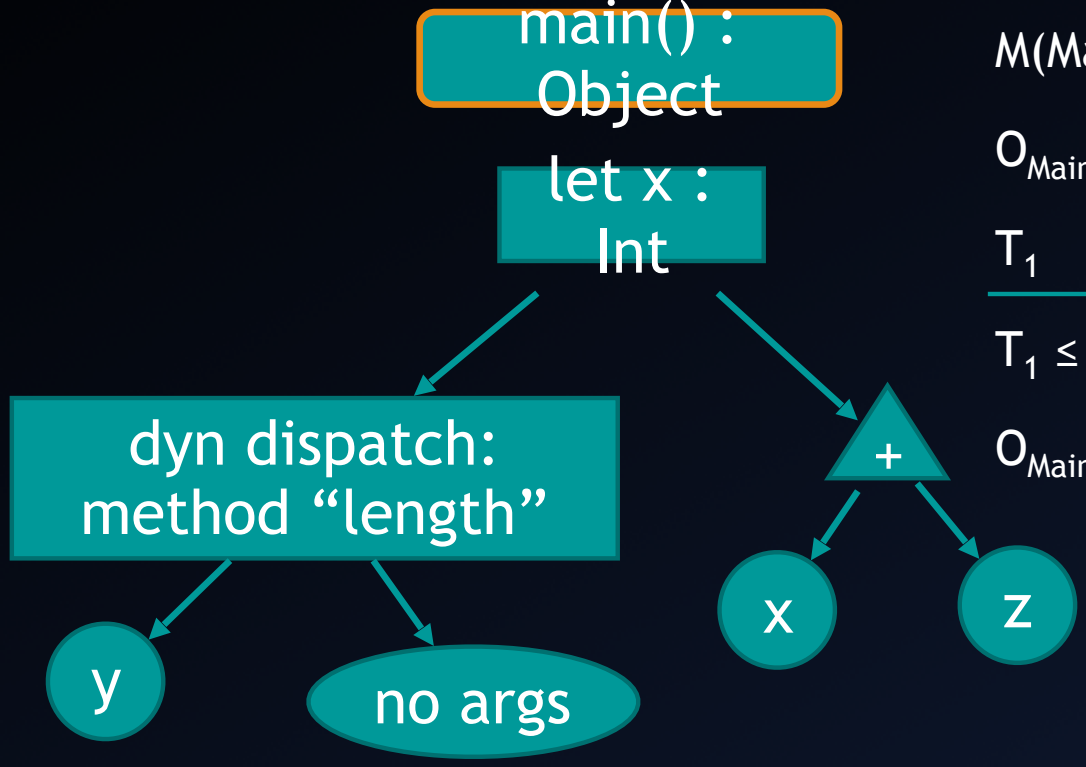
$O_{Main}[SELF\_TYPE_C/self], M, Main \vdash \{ let\ x \ldots \} : T_1$

$T_1 \leq Object$

$O_{Main}, M, Main \vdash main() : T_0 \{ let\ x \ \ldots \}$

Id	Type
y	String
z	Int

Typechecking Example



[Method]

$M(Main,main) = (T_0)$

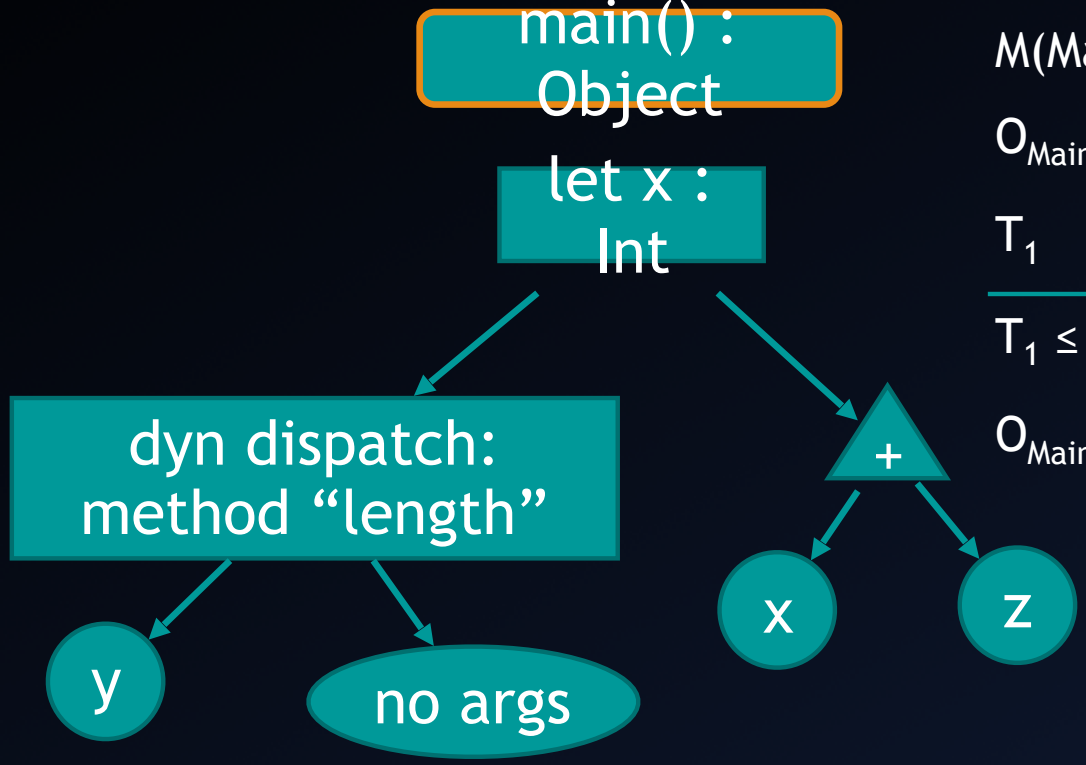
$O_{Main}[SELF\_TYPE_C/self], M, Main \vdash \{ let\ x \ldots \} : T_1$

$T_1 \leq Object$

$O_{Main}, M, Main \vdash main() : T_0 \{ let\ x \ \ldots \}$

Id	Type
y	String
z	Int

Typechecking Example



[Method]

$M(Main,main) = (Object)$

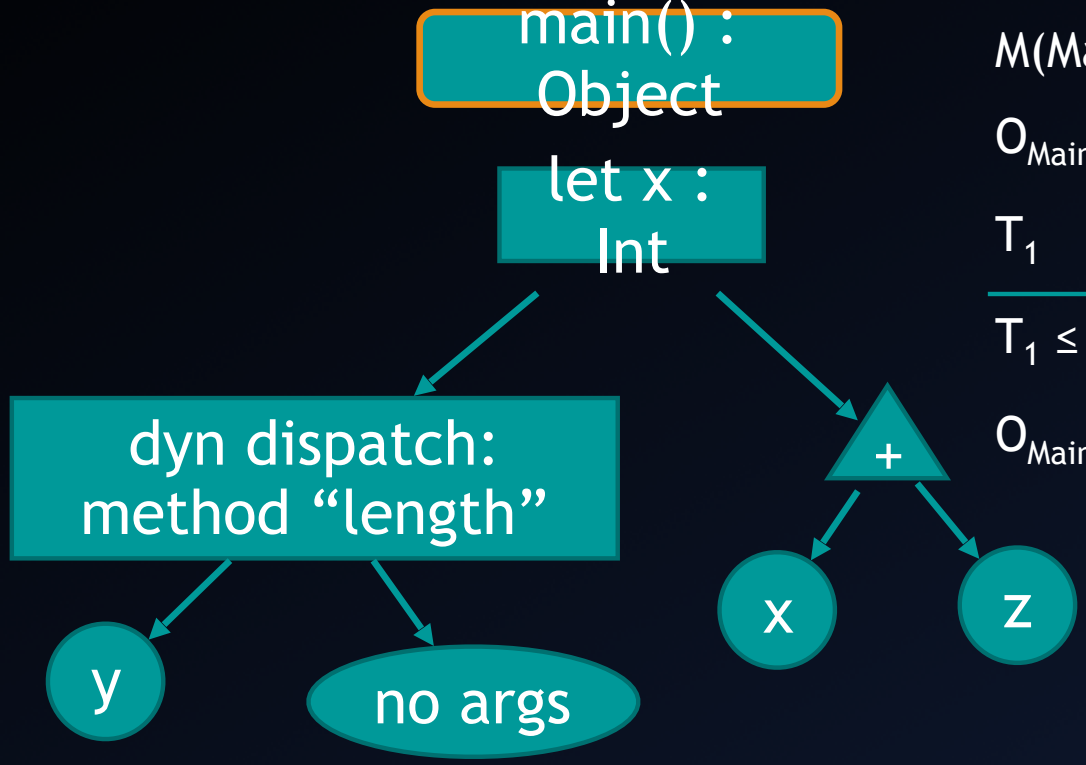
$O_{Main}[SELF\_TYPE_C/self], M, Main \vdash \{ let\ x \ldots \} : T_1$

$T_1 \leq Object$

$O_{Main}, M, Main \vdash main() : Object \{ let\ x \ \ldots \}$

Id	Type
y	String
z	Int

Typechecking Example



[Method]

$M(Main,main) = (Object)$ ✓

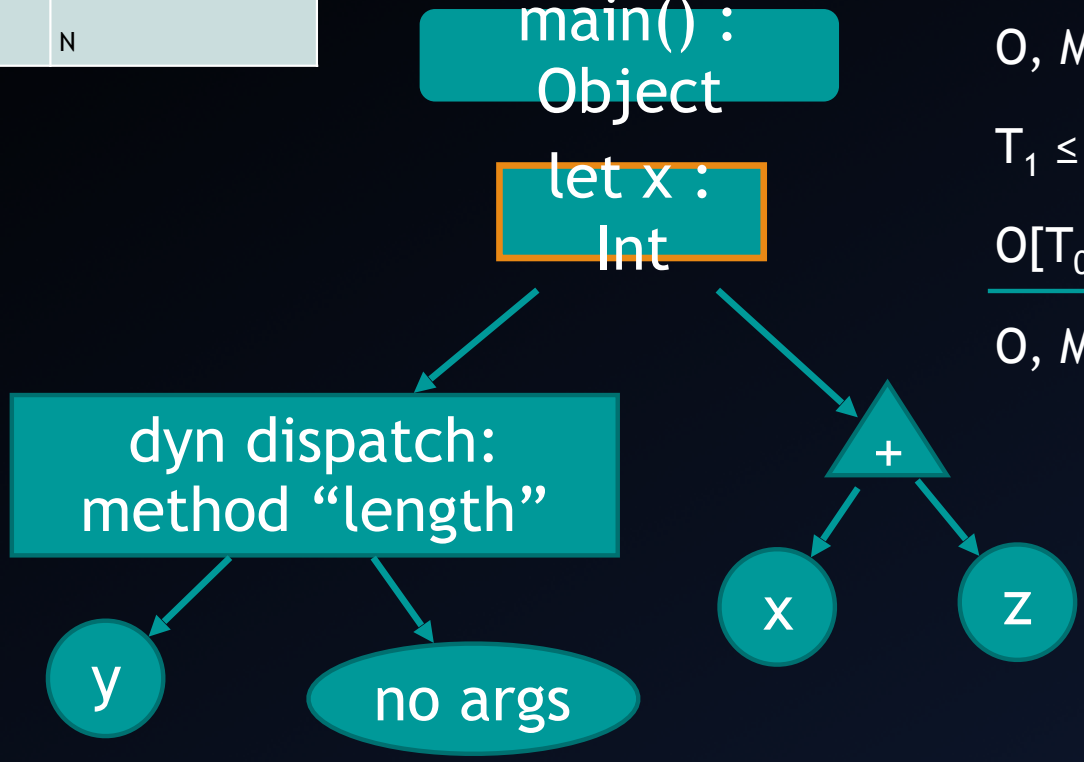
$O_{Main}[SELF\_TYPE_C/self], M, Main \vdash \{ let\ x \ldots \} : T_1$

$T_1 \leq Object$

$O_{Main}, M, Main \vdash main() : Object \{ let\ x \ \ldots \}$

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



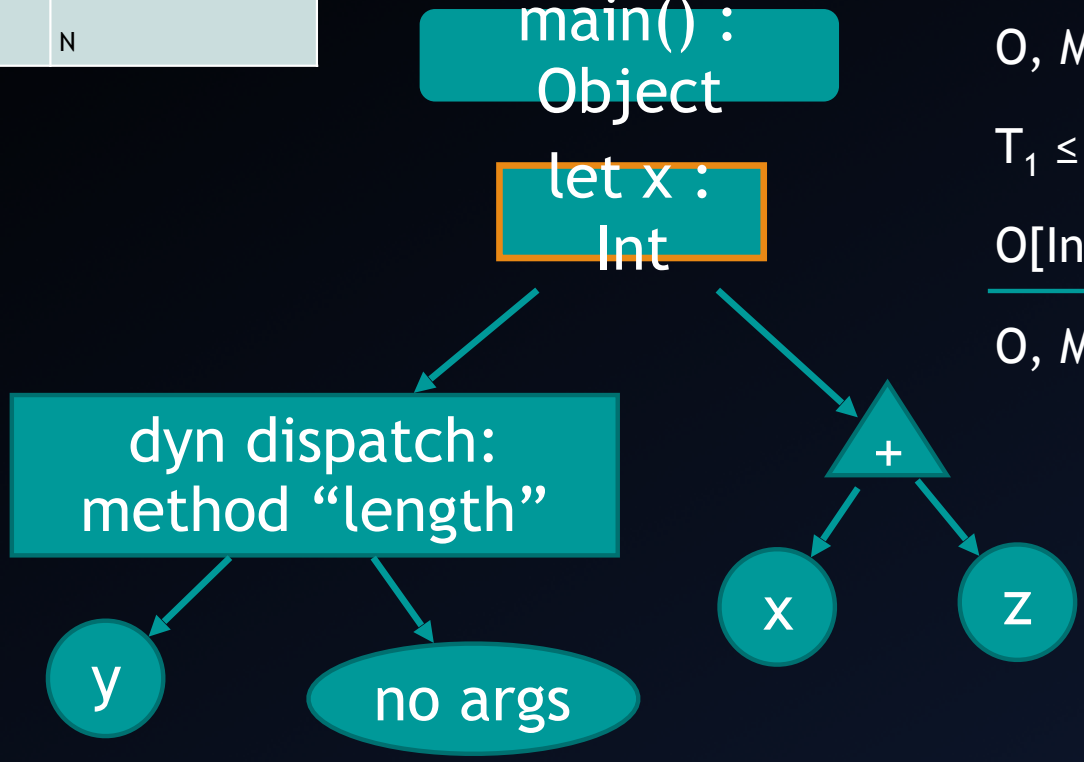
[Let-Init]

$$\frac{O, M, C \vdash e_1 : T_1 \quad T_1 \le T_0 \quad O[T_0/x], M, C \vdash e_2 : T_2}{O, M, C \vdash \text{let } x : T_0 \leftarrow e_1 \text{ in } e_2 : T_2}$$

e_1 -> y.length()
e_2 -> x + z
T_0 -> Int

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$

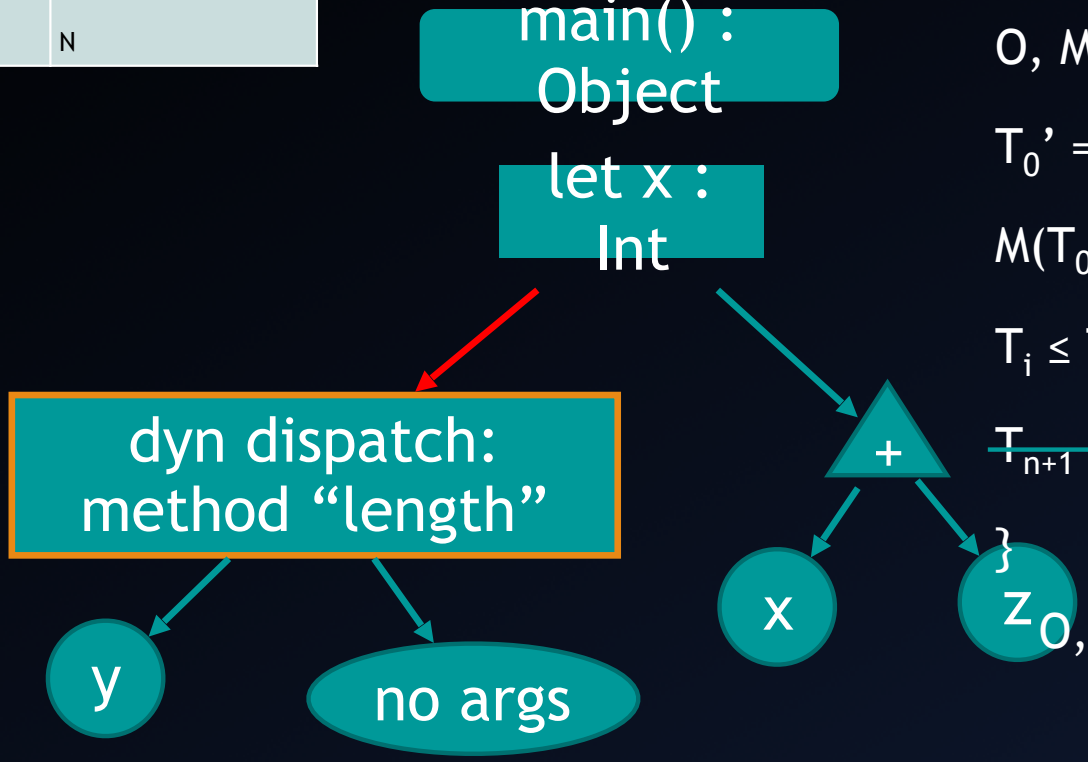


[Let-Init]

$$\frac{O, M, C \vdash y.length() : T_1 \quad T_1 \leq Int \quad O[Int/x], M, C \vdash x+z : T_2}{O, M, C \vdash let\ x : Int \leftarrow y.length()\ in\ x+z : T_2}$$

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Dispatch]

$O, M, C \vdash e_0 : T_0 \quad \ldots \quad O, M, C \vdash e_n : T_n$

$T_0' = \{ C \text{ if } T_0 = SELF\_TYPE_C;\ T_0 \text{ otherwise } \}$

$M(T_0', f) = (T_1', \ldots, T_{n+1}')$

$T_i \le T_i' \quad \text{for} \quad 1 \le i \le n$

~~$T_{n+1} = \{ T_0 \text{ if } T'_{n+1} = SELF\_TYPE;\ T'_{n+1} \text{ otherwise } \}$~~

$O, M, C \vdash e_0.f(e_1, \ldots, e_n) : T_{n+1}$

n -> 0 (there are no arguments)

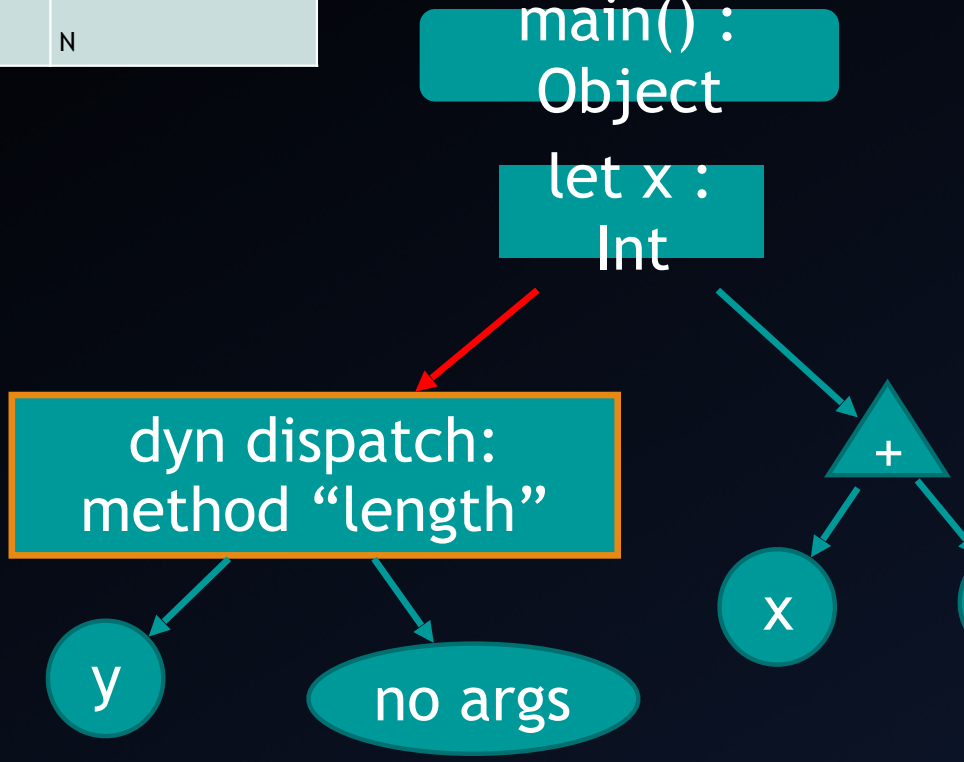
f -> “length”

e_0 -> y

C -> Main

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Dispatch]

$O, M, Main \vdash e_0 : T_0$

$T_0' = \{ Main \text{ if } T_0 = SELF\_TYPE_{Main};\ T_0 \text{ otherwise } \}$

$M(T_0', \text{"length"}) = (T_1')$

$T_1 \le T_1'$

$T_1 = \{ T_0 \text{ if } T'_1 = SELF\_TYPE; T'_1 \text{ otherwise } \}$

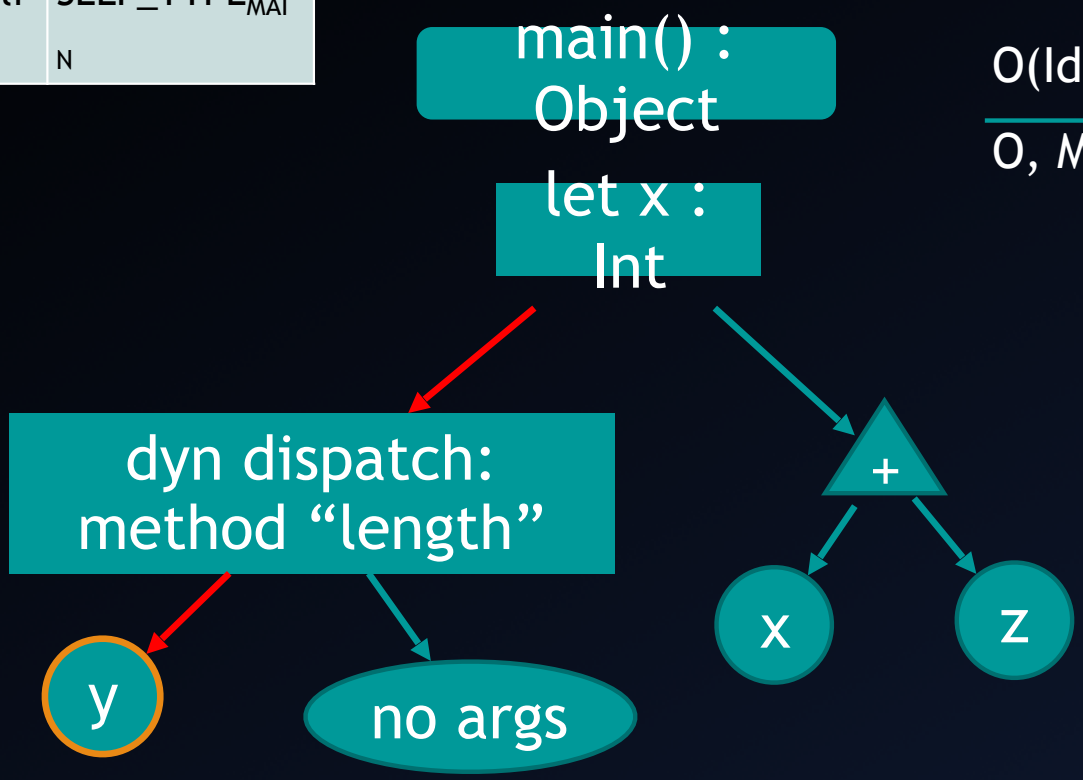
$O, M, Main \vdash y.length() : T_1$

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$

[Var]

$$\frac{O(Id) = T}{O, M, C \vdash Id : T}$$

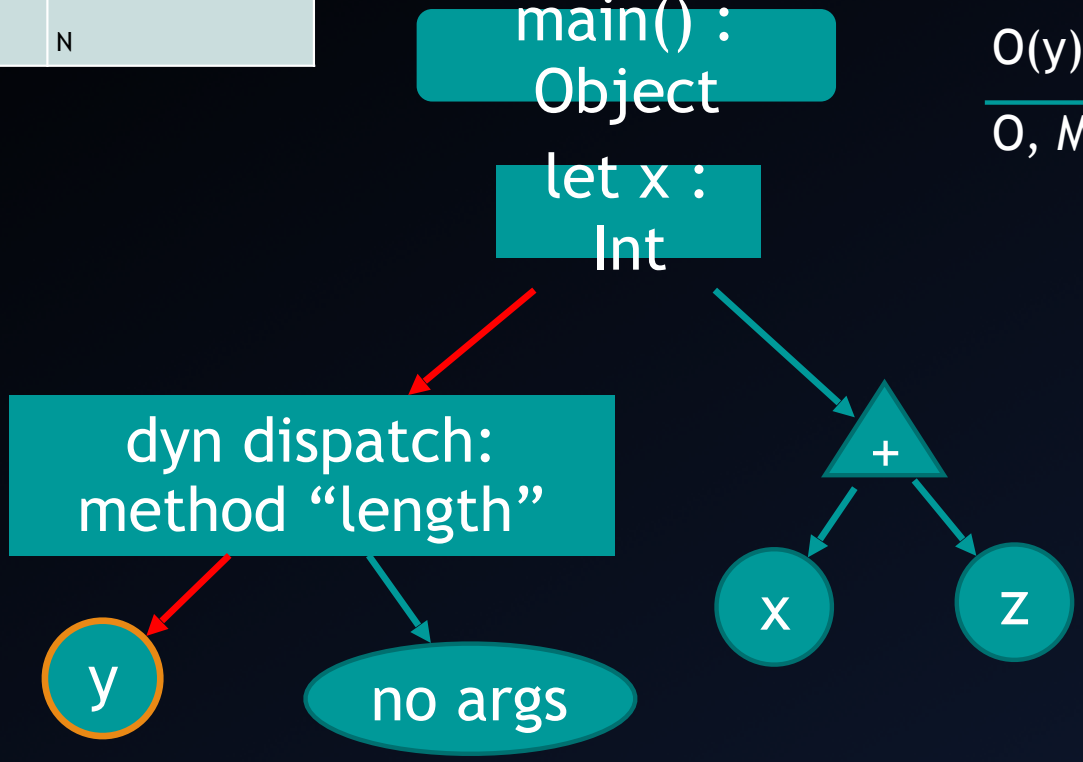


Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$

[Var]

$$\frac{O(y) = String \checkmark}{O, M, C \vdash y : String \checkmark}$$

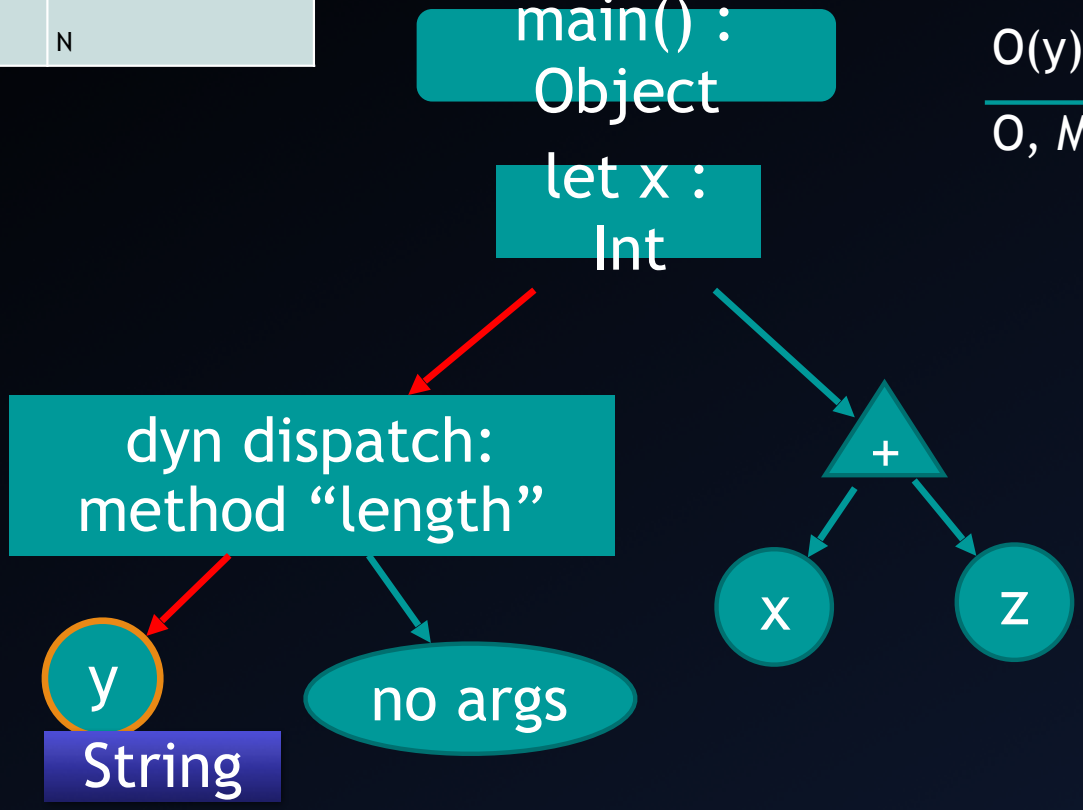


Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$

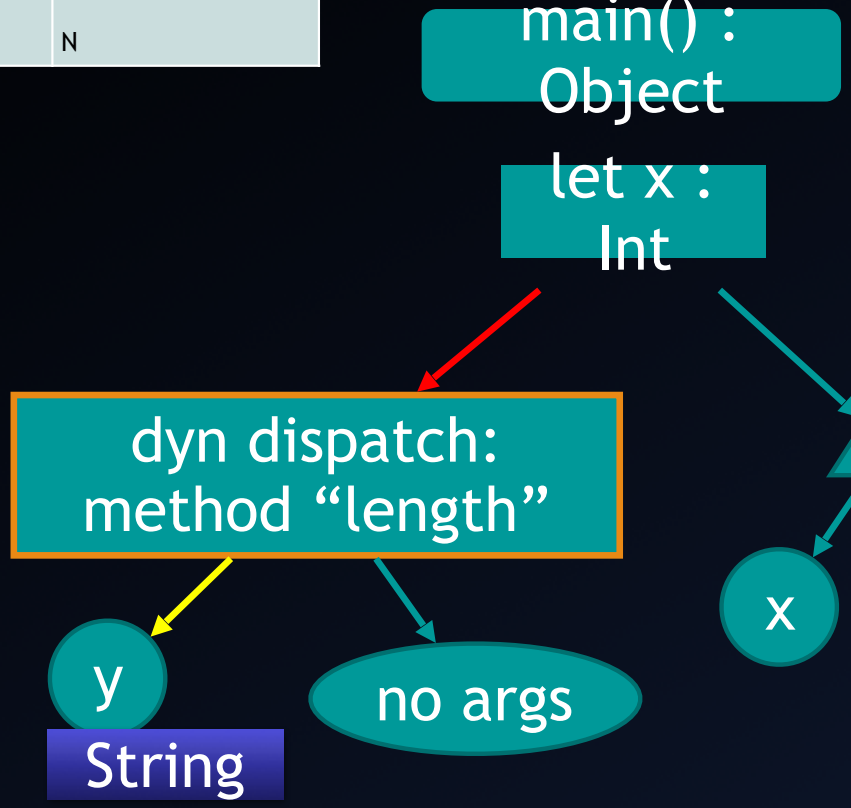
[Var]

$$\frac{O(y) = String \checkmark}{O, M, C \vdash y : String \checkmark}$$



Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Dispatch]

$O, M, C \vdash y : T_0$

$T_0' = \{ Main \text{ if } T_0 = SELF\_TYPE_{Main};\ T_0 \text{ otherwise } \}$

$M(T_0', \text{"length"}) = (T_1')$

$T_1 \le T_1'$

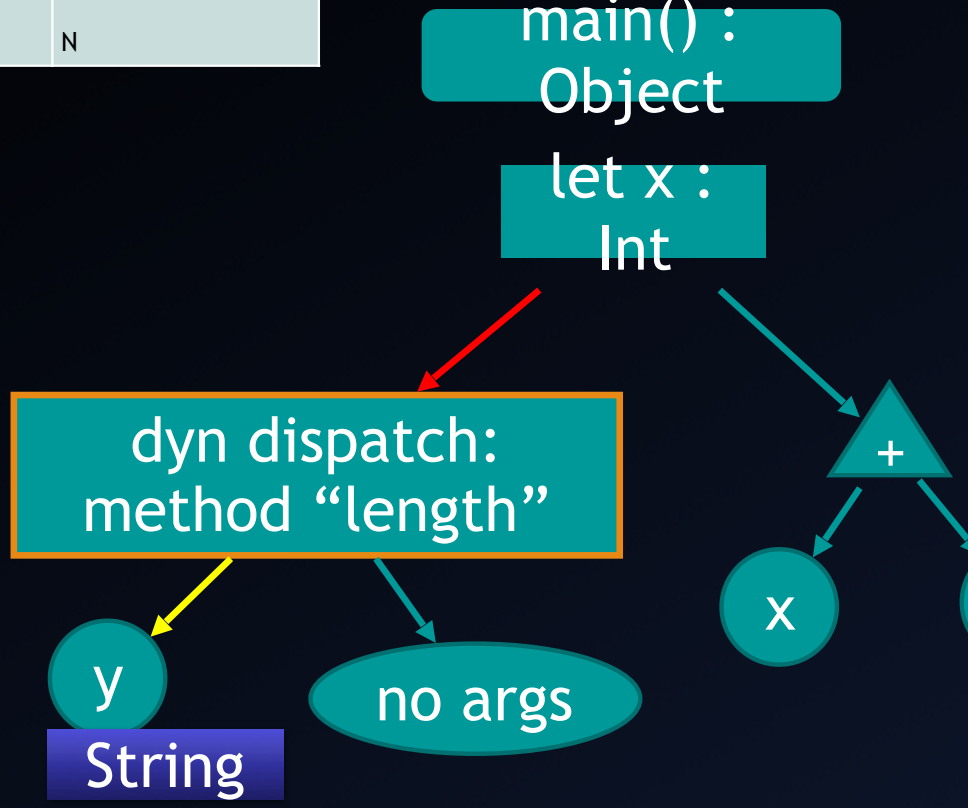
$T_1 = \{ T_0 \text{ if } T'_1 = SELF\_TYPE; T'_1 \text{ otherwise } \}$

$O, M, C \vdash y.length() : T_1$

$T_0 \rightarrow String$

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Dispatch]

$O, M, C \vdash y : String$ ✓

$T_0' = \{ Main \text{ if } Str = SELF\_TYPE_{Main};\ Str$

$\text{otherwise} \}$

$M(T_0', \text{"length"}) = (T_1')$

$T_1 \le T_1'$

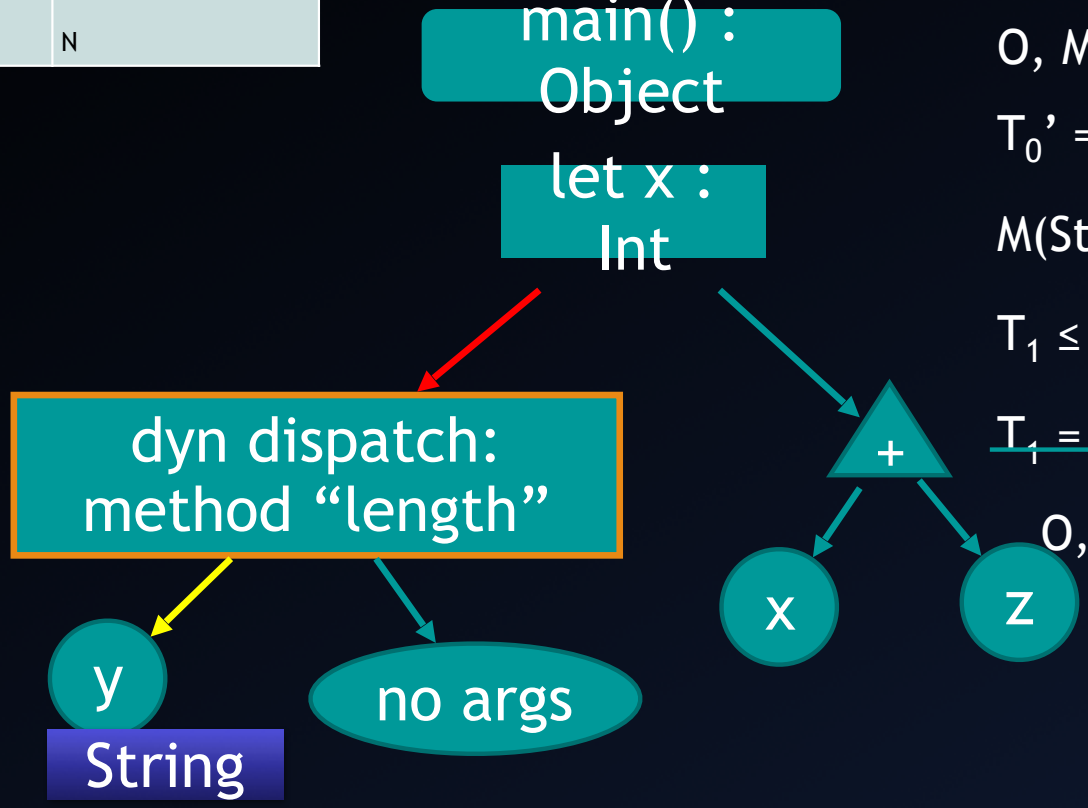
$T_1 = \{ Str \text{ if } T'_1 = SELF\_TYPE;\ T'_1 \text{ otherwise} \}$

$O, M, C \vdash y.length() : T_1$

Now $T_0' = String$

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Dispatch]

$O, M, C \vdash y : String$ ✓

$T_0' = String$ ✓

$M(String, \text{"length"}) = (T_1')$

$T_1 \le T_1'$

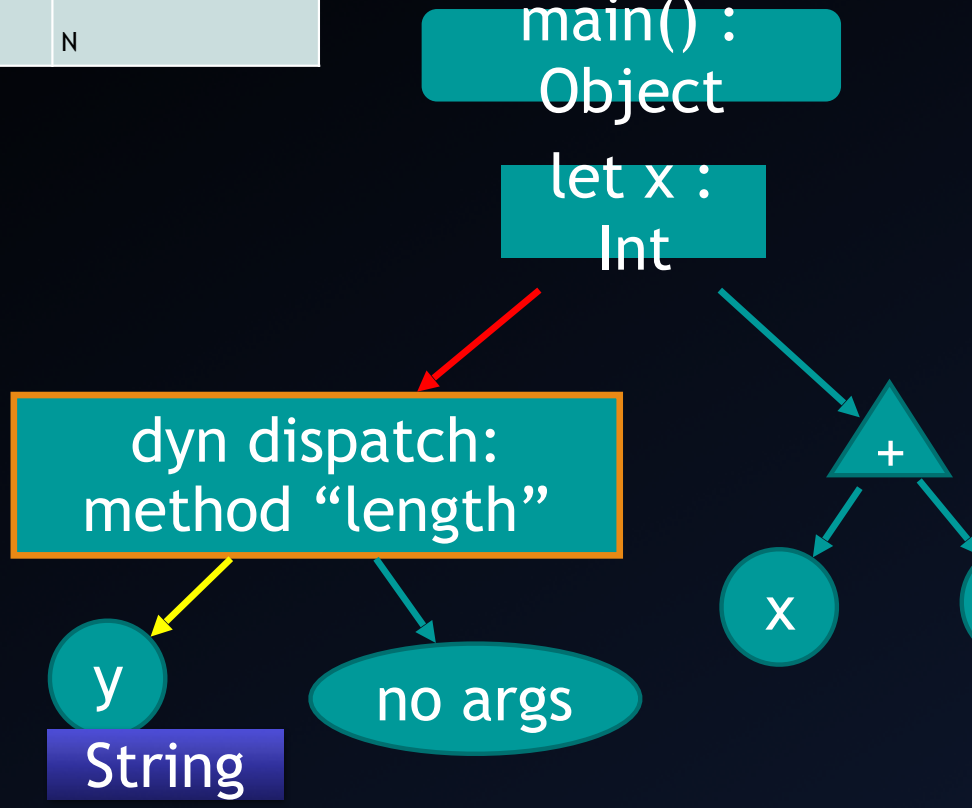
~~$T_1 = \{ Str \text{ if } T'_1 = SELF\_TYPE; T'_1 \text{ otherwise} \}$~~

$O, M, C \vdash y.length() : T_1$

Now, look up String.length in the method environment. Does it really have no parameters?

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Dispatch]

$O, M, C \vdash y : String$ ✓

$T_0' = String$ ✓

$M(String, \text{"length"}) = (T_1') = (Int)$

$T_1 \le T_1'$

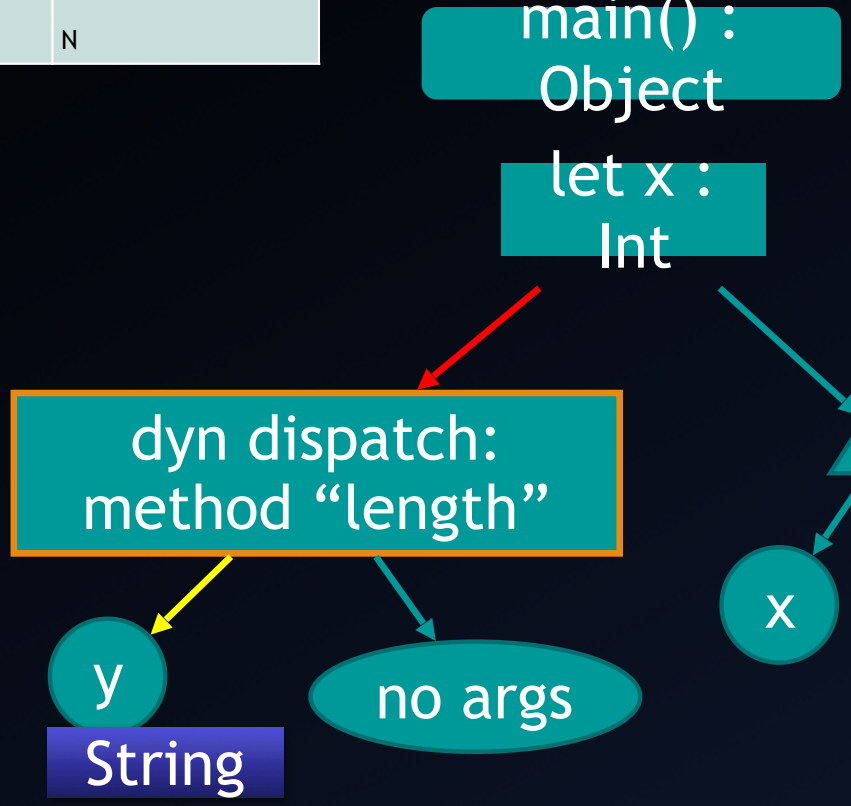
~~$T_1 = \{ Str \text{ if } T'_1 = SELF\_TYPE; T'_1 \text{ otherwise} \}$~~

$O, M, C \vdash y.length() : T_1$

Yes! It returns an Int.
Now T1 = T1' = Int.

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Dispatch]

$O, M, C \vdash y : String$ ✓

$T_0' = String$ ✓

$M(String, \text{“length”}) = (T_1') = (Int')$ ✓

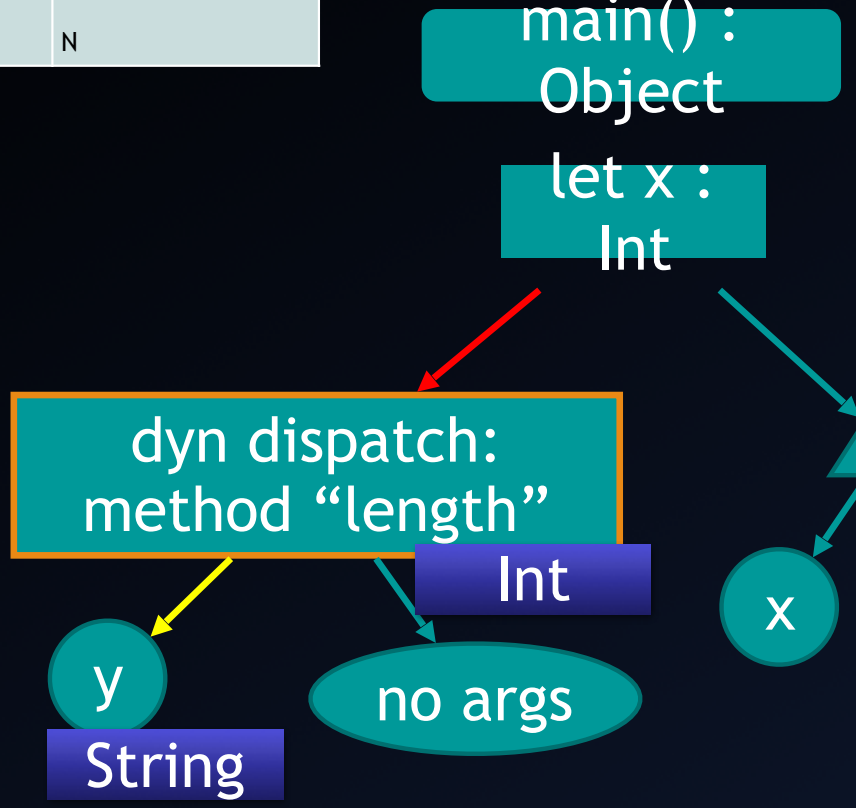
$T_1 \le T_1' \Leftrightarrow Int \le Int'$ ✓

$T_1 = Int$ ✓

$O, M, C \vdash y.length() : Int$

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Dispatch]

$O, M, C \vdash y : String$ ✓

$T_0' = String$ ✓

$M(String, \text{"length"}) = (T_1') = (Int')$ ✓

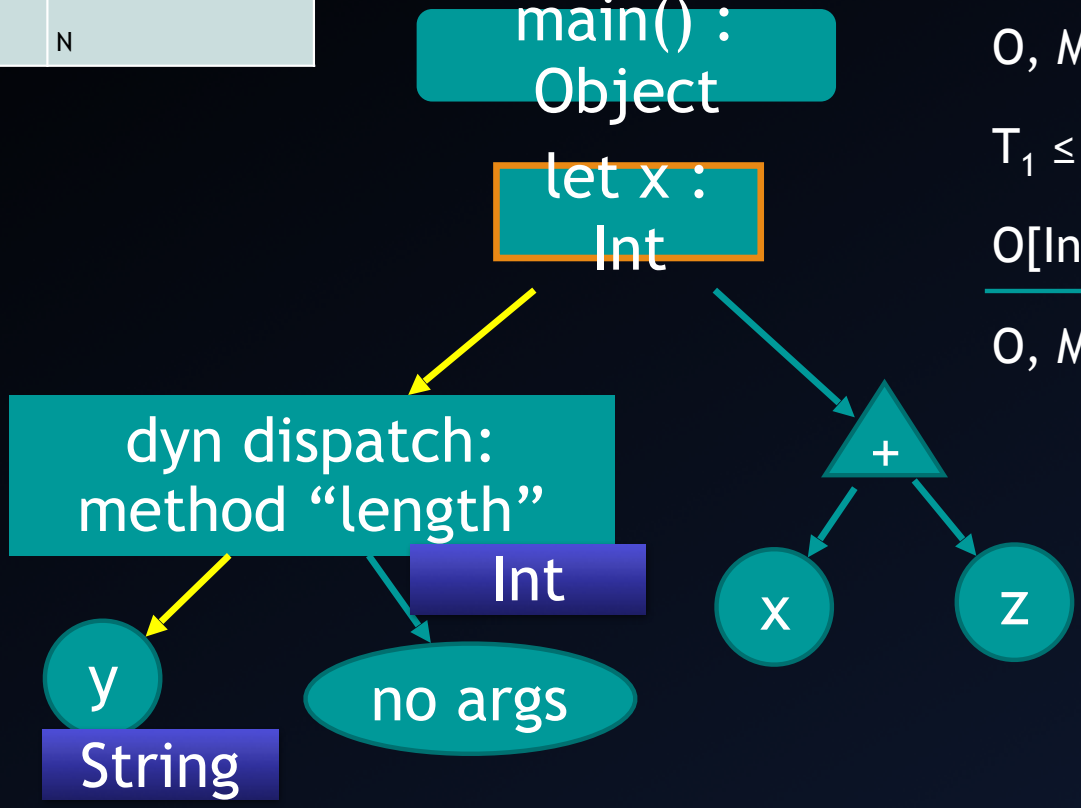
$T_1 \le T_1' \Leftrightarrow Int \le Int'$ ✓

$T_1 = Int$ ✓

$O, M, C \vdash y.length() : Int$ ✓

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Let-Init]

$O, M, C \vdash y.length() : T_1$

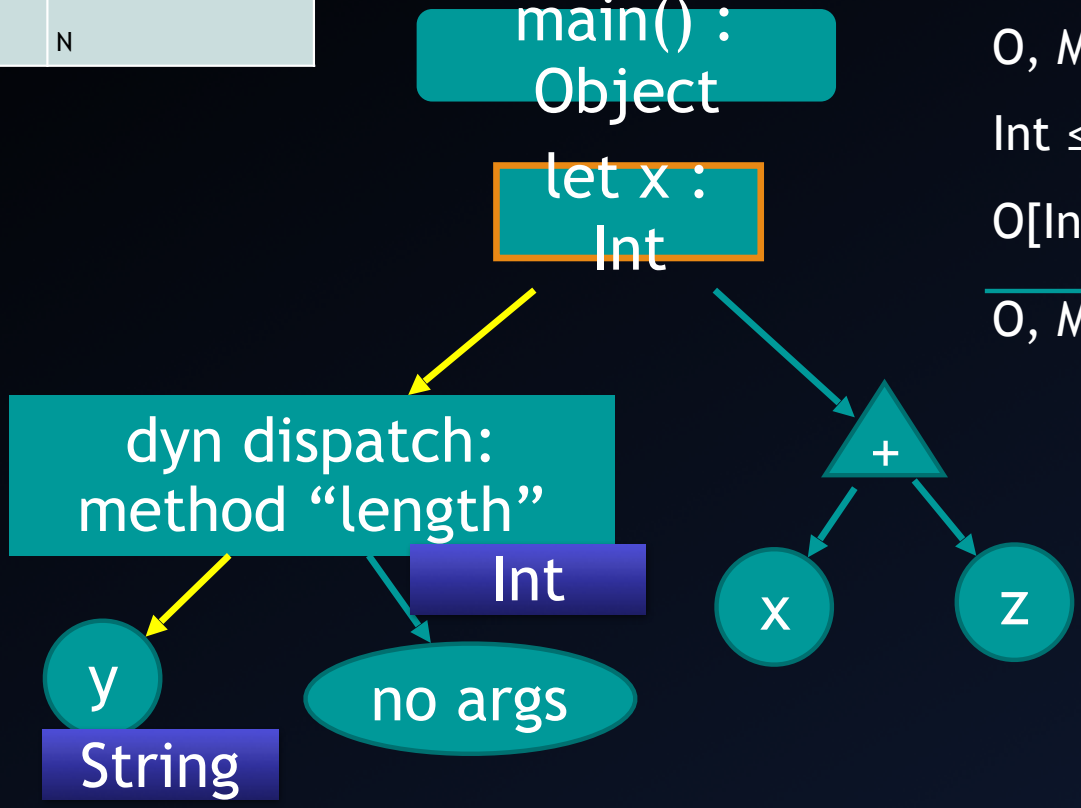
$T_1 \leq Int$

$O[Int/x], M, C \vdash x+z : T_2$

$O, M, C \vdash let\ x : Int \leftarrow y.length()\ in\ x+z : T_2$

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Let-Init]

$O, M, C \vdash y.length() : Int$ ✓

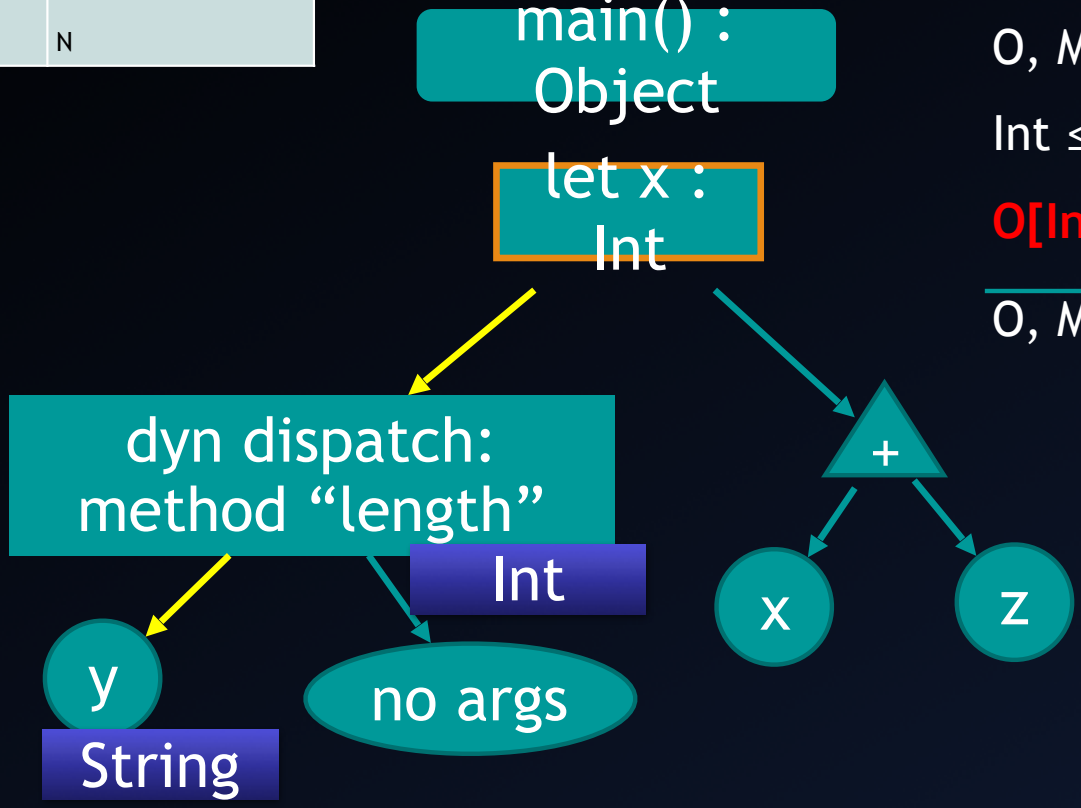
$Int \leq Int$ ✓

$O[Int/x], M, C \vdash x+z : T_2$

$O, M, C \vdash let\ x : Int \leftarrow y.length()\ in\ x+z : T_2$

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Let-Init]

$O, M, C \vdash y.length() : Int$ ✓

$Int \leq Int$ ✓

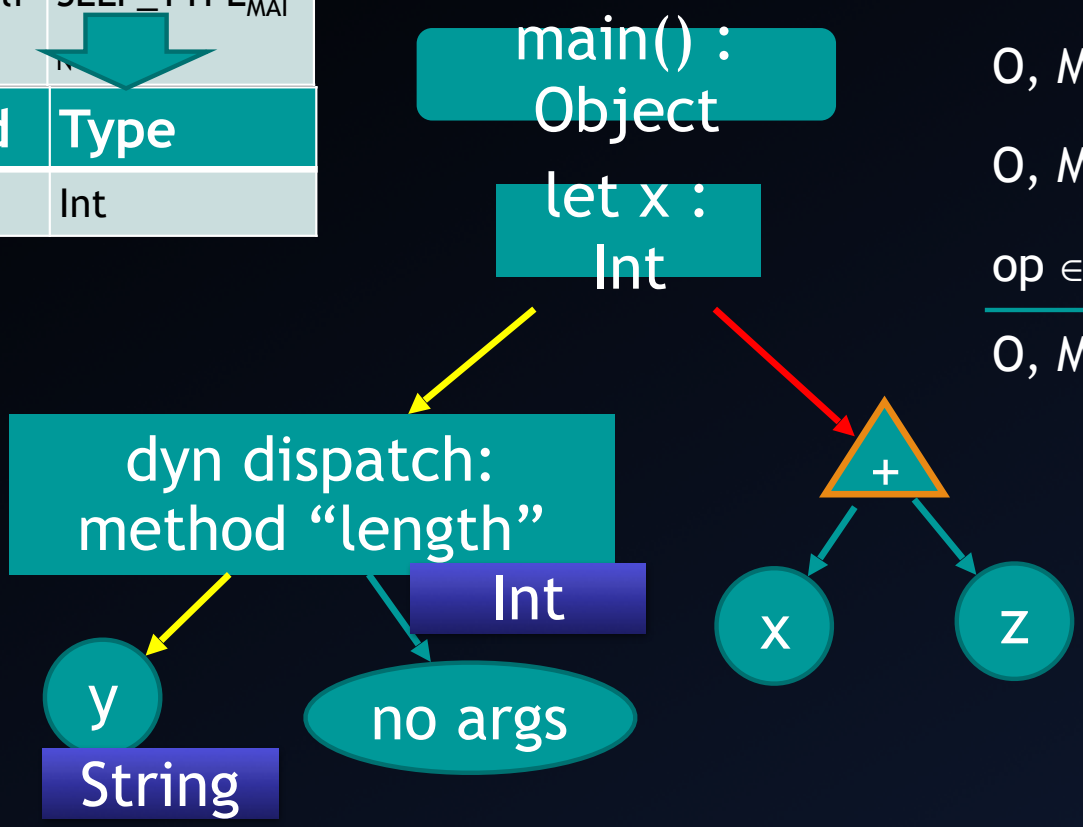
$O[Int/x], M, C \vdash x+z : T_2$

$O, M, C \vdash let\ x : Int \leftarrow y.length()\ in\ x+z : T_2$

Typechecking Example

Id	Type
y	String
z	Int
self	SELF_TYPE$_{MAI}$

Id	Type
x	Int



[Arith]

$$\frac{O, M, C \vdash e_1 : \text{Int} \quad O, M, C \vdash e_2 : \text{Int} \quad \text{op} \in \{ *, +, -, / \}}{O, M, C \vdash e_1 \text{ op } e_2 : \text{Int}}$$

e_1 -> x

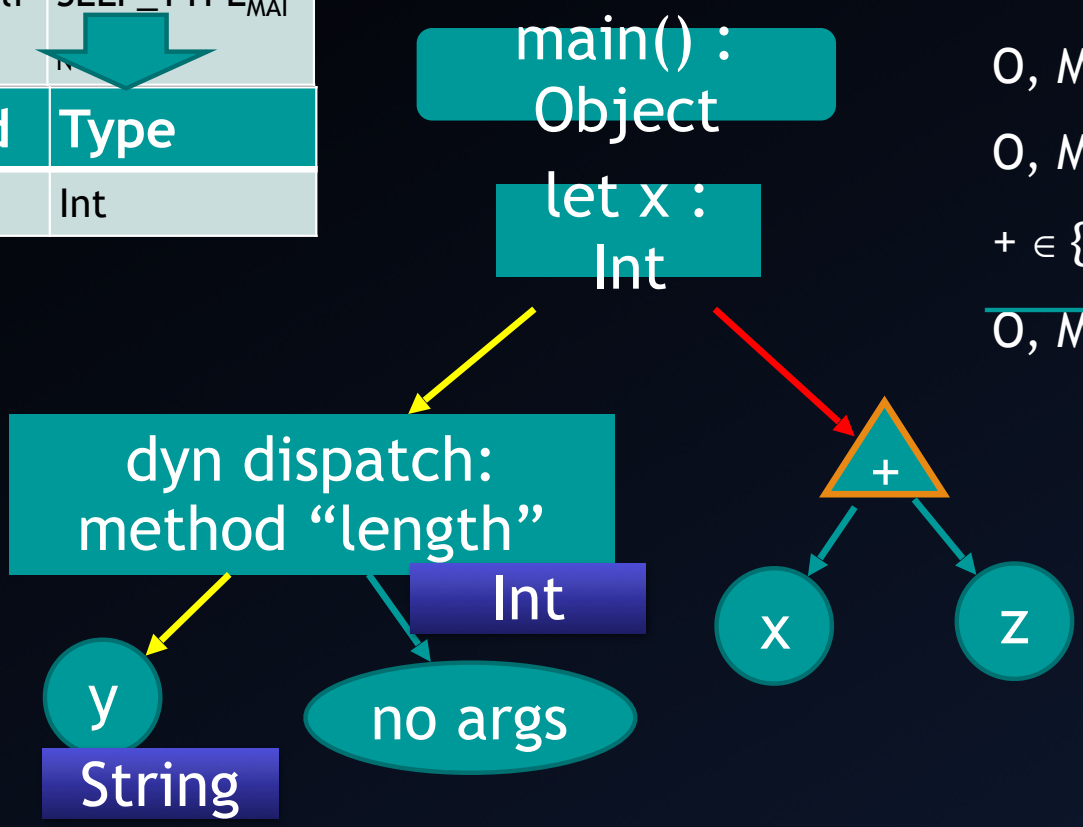
e_2 -> z

op -> +

Typechecking Example

Id	Type
y	String
z	Int
self	SELF_TYPE$_{MAI}$

Id	Type
x	Int



[Arith]

$$\frac{O, M, C \vdash x : \text{Int} \quad O, M, C \vdash z : \text{Int} \quad + \in \{ *, +, -, / \} \checkmark}{O, M, C \vdash x + z : \text{Int}}$$

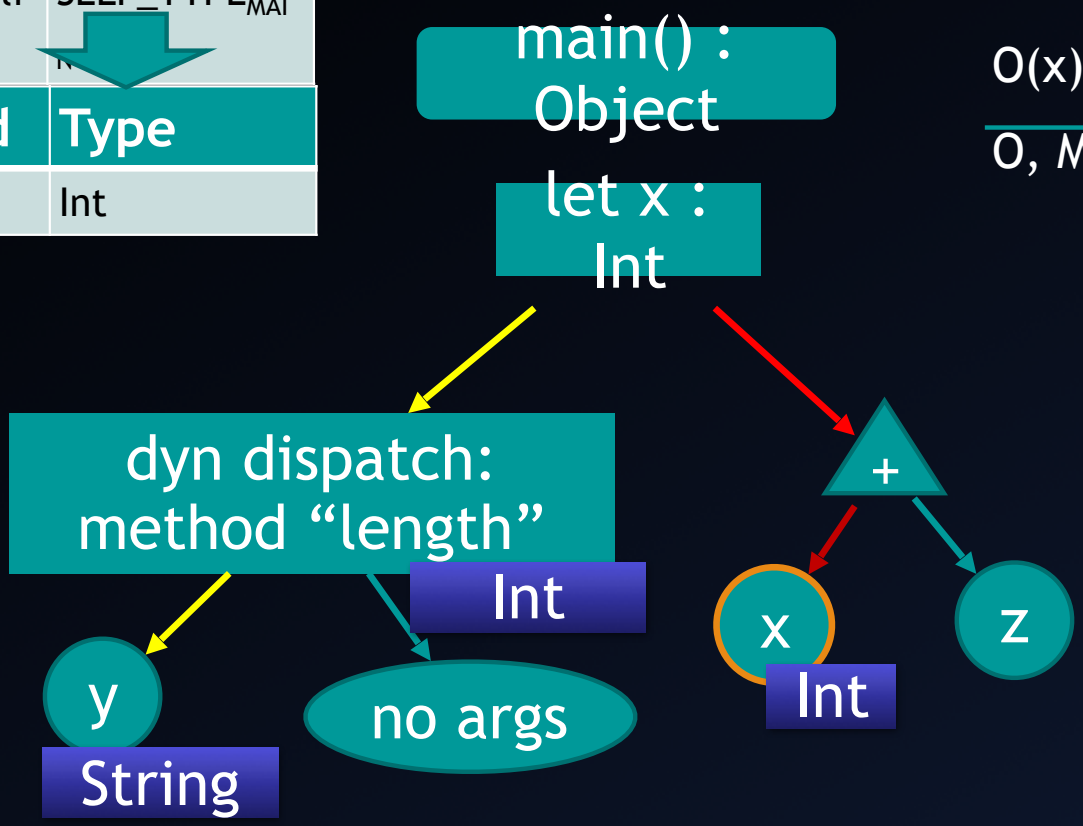
Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAI}$

Id	Type
x	Int

[Var]

$$\frac{O(x) = Int \quad \checkmark}{O, M, C \vdash y : Int \quad \checkmark}$$



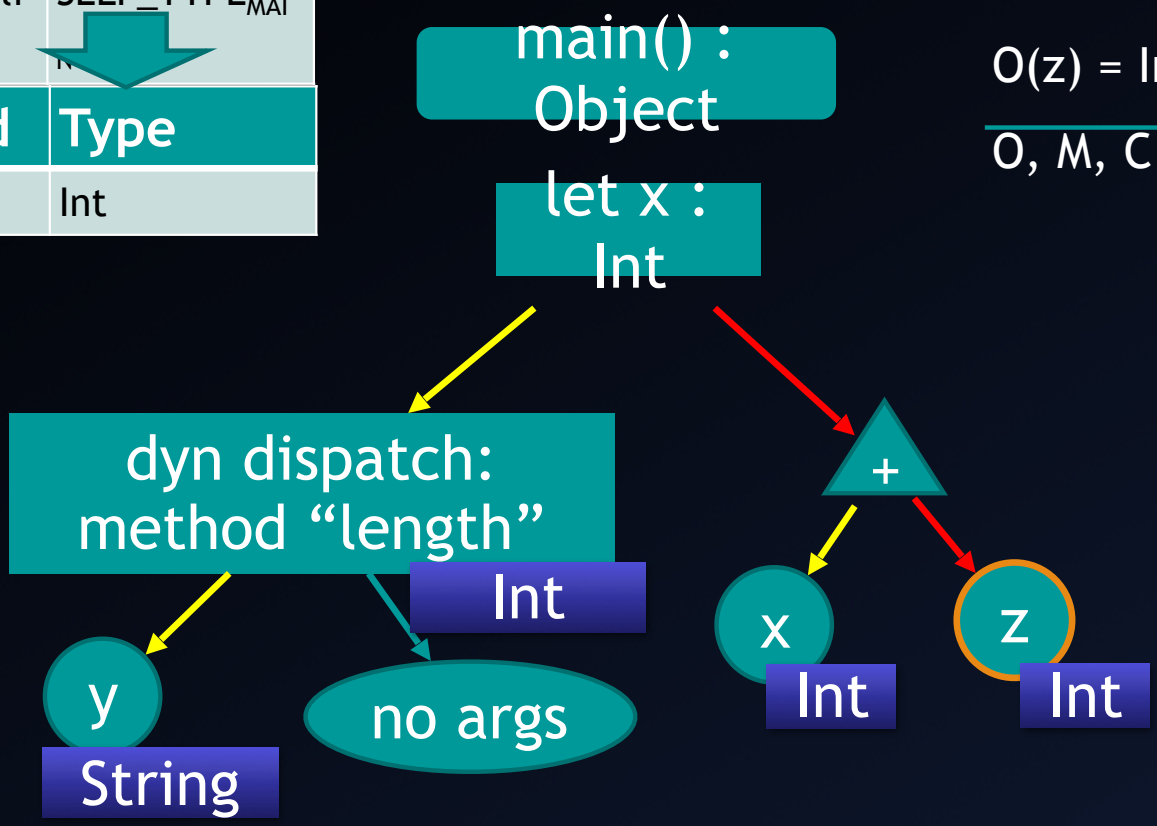
Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAI}$

Id	Type
x	Int

[Var]

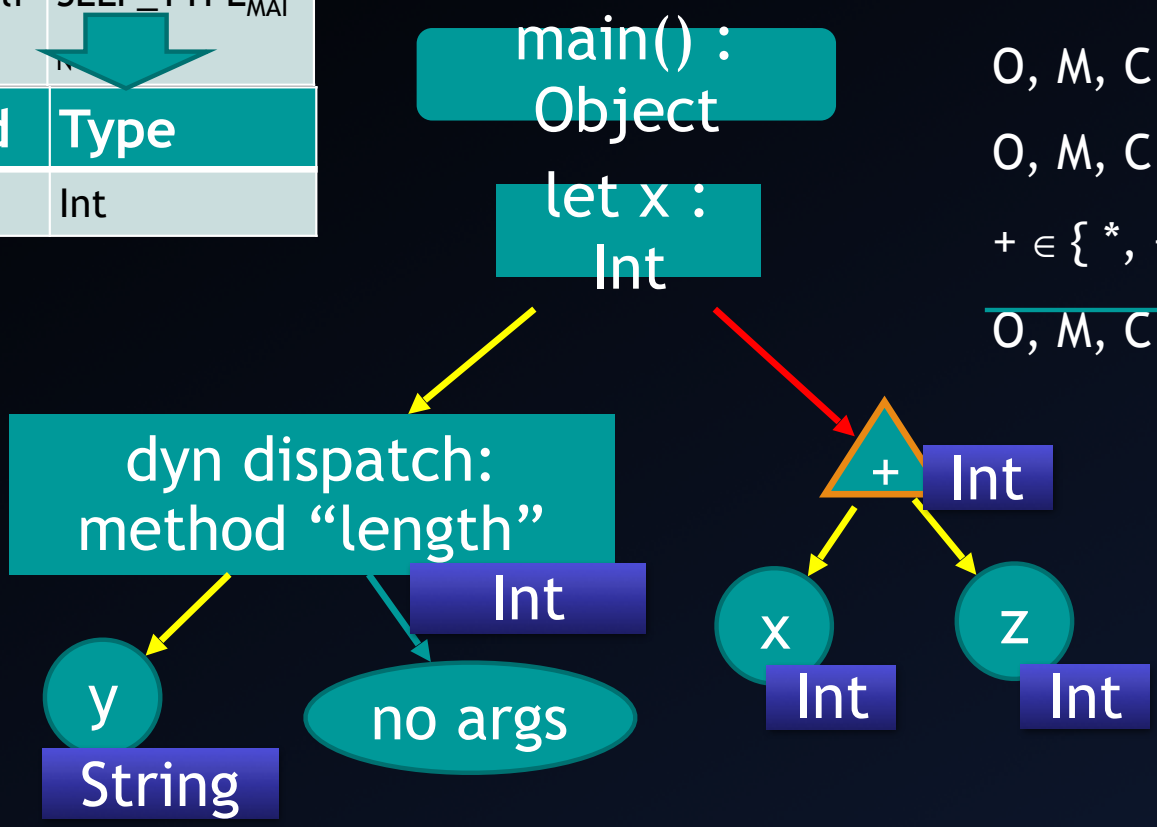
$$\frac{O(z) = Int \quad \checkmark}{O, M, C \vdash z : Int \quad \checkmark}$$



Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAI}$

Id	Type
x	Int



[Arith]

$O, M, C \vdash x : Int$ ✓

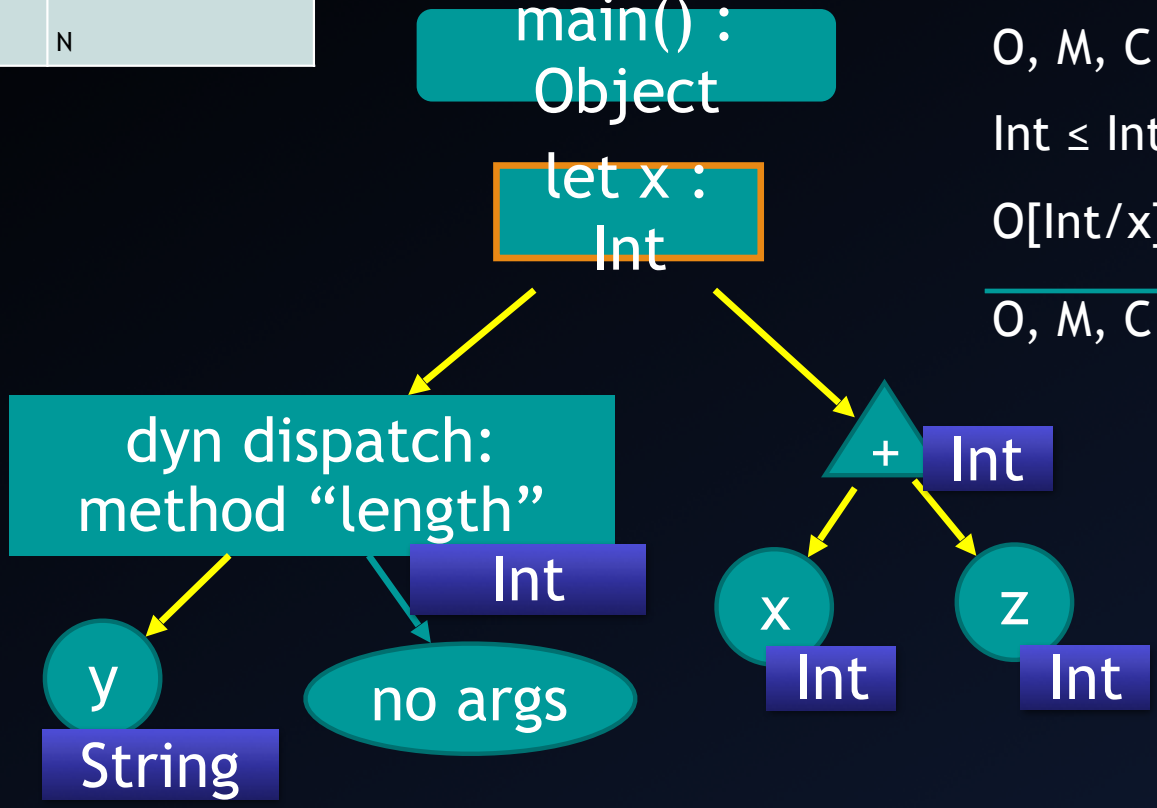
$O, M, C \vdash z : Int$ ✓

$+ \in \{ *, +, -, / \}$ ✓

$O, M, C \vdash x + z : Int$ ✓

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Let-Init]

$O, M, C \vdash y.length() : Int$ ✓

$Int \leq Int$ ✓

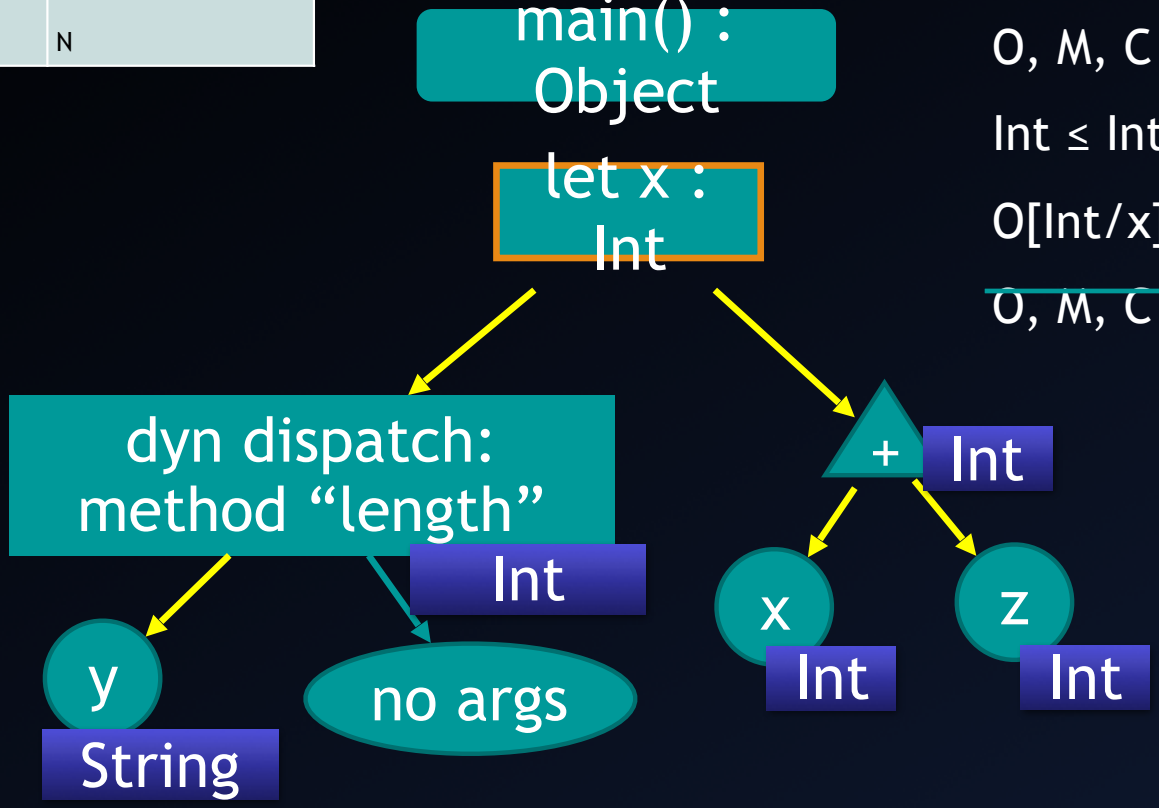
$O[Int/x], M, C \vdash x+z : T_2$

$O, M, C \vdash let\ x : Int \leftarrow y.length()\ in\ x+z : T_2$

T_2 -> Int

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Let-Init]

$O, M, C \vdash y.length() : Int$ ✓

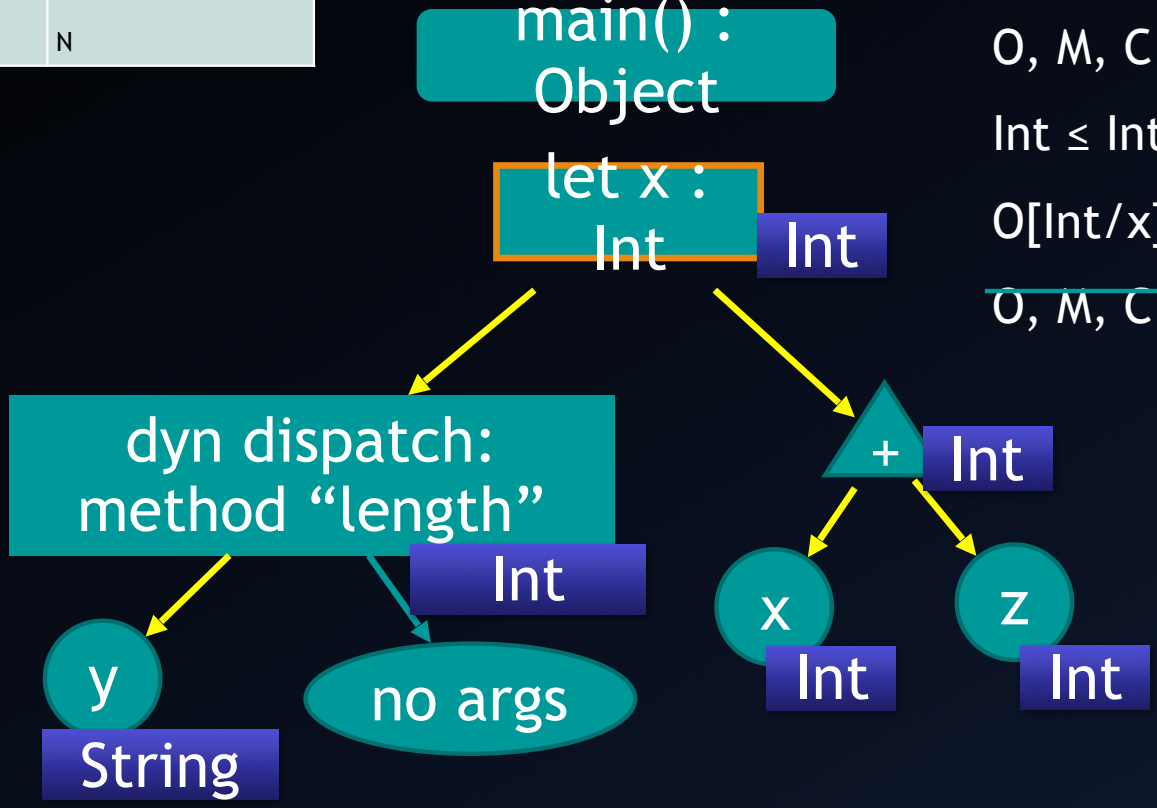
$Int \leq Int$ ✓

$O[Int/x], M, C \vdash x+z : Int$ ✓

$O, M, C \vdash let\ x : Int \leftarrow y.length()\ in\ x+z : Int$ ✓

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Let-Init]

$O, M, C \vdash y.length() : Int$ ✓

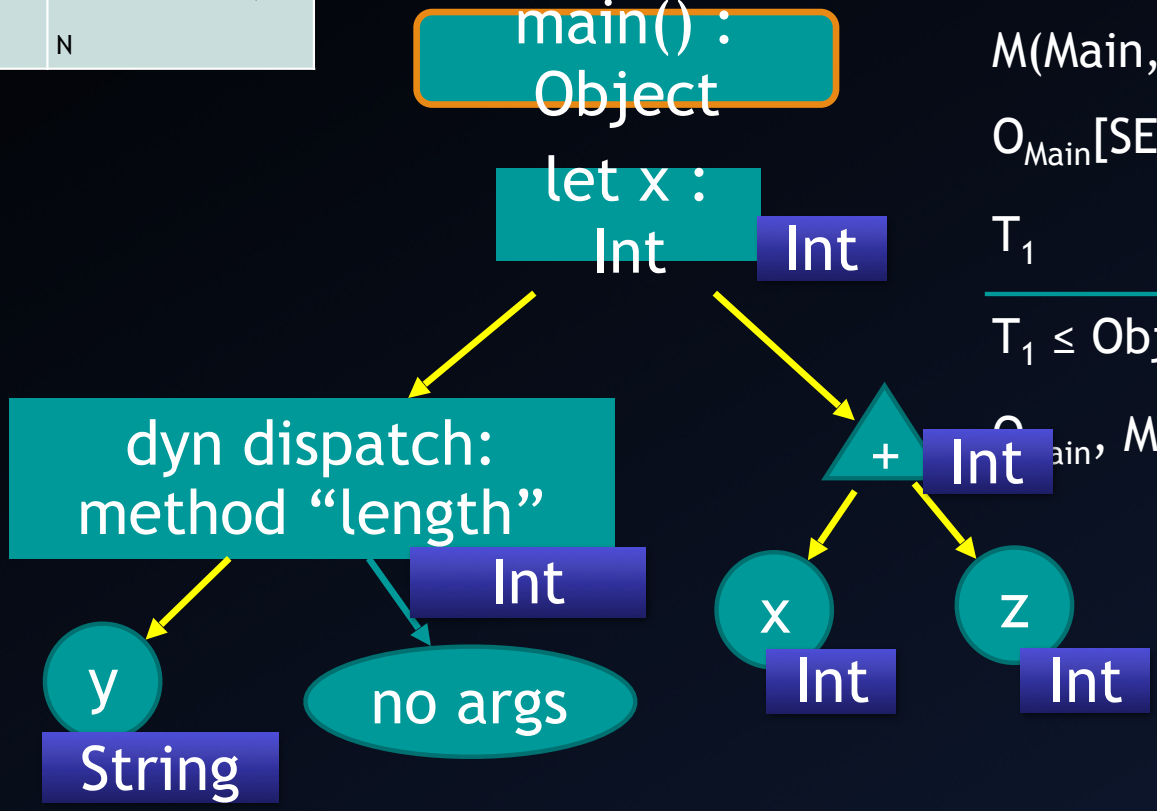
$Int \leq Int$ ✓

$O[Int/x], M, C \vdash x+z : Int$ ✓

$O, M, C \vdash let\ x : Int \leftarrow y.length()\ in\ x+z : Int$ ✓

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Method]

$M(Main, main) = (Object)$ ✓

$O_{Main}[SELF\_TYPE_C/self], M, Main \vdash \{ let\ x \ldots \} : T_1$

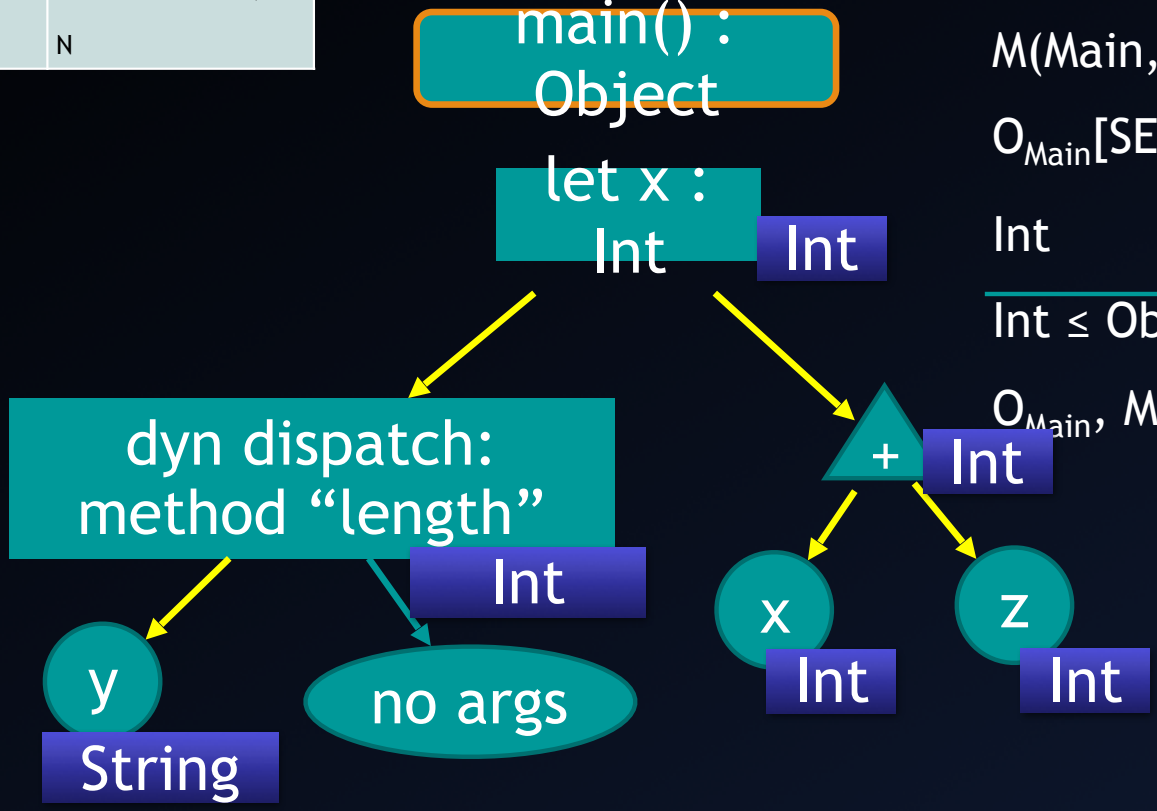
$T_1 \leq Object$

$O_{Main}, M, Main \vdash main() : Object \{ let\ x \ldots \}$

$T_1 \rightarrow Int$

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Method]

$M(Main, main) = (Object)$ ✓

$O_{Main}[SELF\_TYPE_C/self], M, Main \vdash \{ let\ x \ldots \} :$ ✓

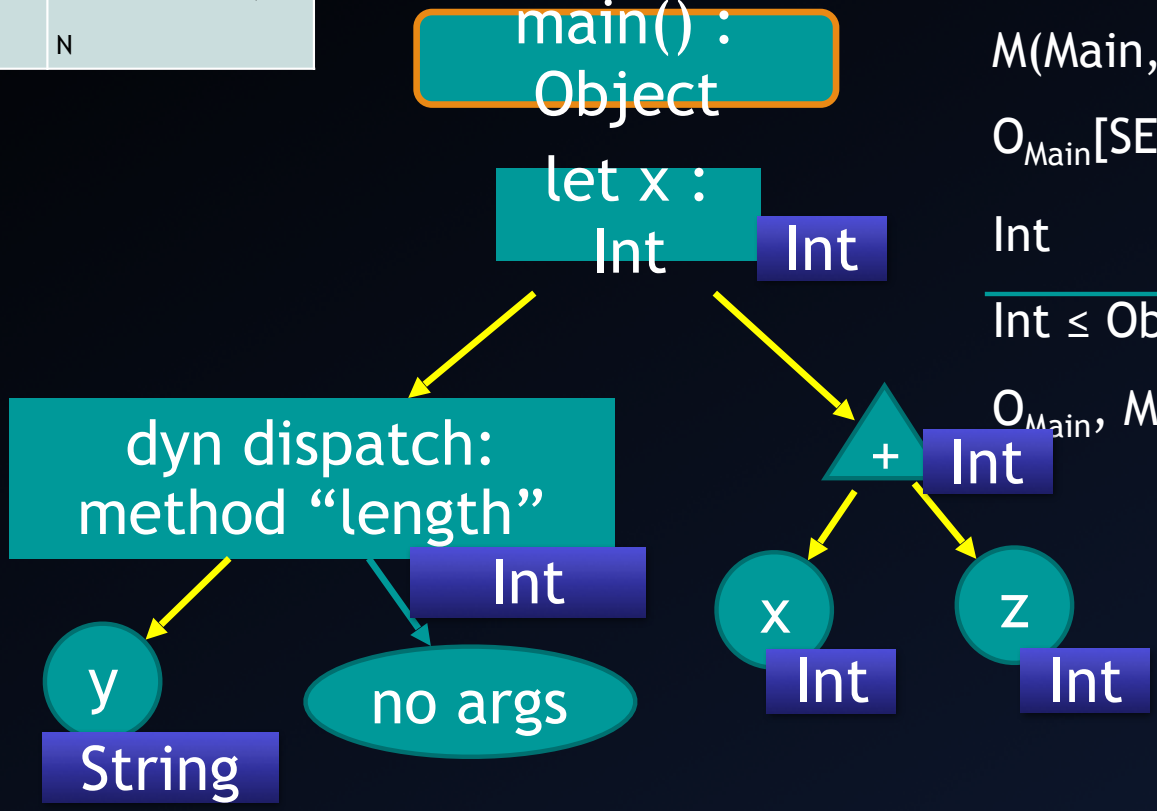
Int ✓

$Int \leq Object$ ✓

$O_{Main}, M, Main \vdash main() : Object \{ let\ x \ \ldots \}$

Typechecking Example

Id	Type
y	String
z	Int
self	$SELF\_TYPE_{MAIN}$



[Method]

$M(Main, main) = (Object)$ ✓

$O_{Main}[SELF\_TYPE_C/self], M, Main \vdash \{ let\ x \ldots \} :$ ✓

Int ✓

$Int \leq Object$ ✓

$O_{Main}, M, Main \vdash main() : Object \{ let\ x \ \ldots \}$