Part I

Objects and Constructors

```
fun intV(n):
  [values(#'apply, fun (arg_val): ....),
   values(#'number, fun (): ....)]

fun closV(n, body, c_env):
  [values(#'apply, fun (arg_val): ....),
   values(#'number, fun (): ....)]
```

- result of `intV` or `closV` is an ***object***
- functions for `#'apply` and `#'number` are ***methods***
- `n`, `body`, and `c_env` are ***fields***
- `intV` and `closV` themselves are ***constructors***

... and not far from ***classes***

Classes

Classes play two (dynamic) roles:

- Object construction

```
class Snake {
  ....
}

new Snake("Slinky", 10);
```

- Implementation inheritance

```
class Rattlesnake extends Snake {
  ....
}
```

  - Inheritance of methods
  - Static method dispatch

Classes: Static and Dynamic Dispatch

```
class Snake implements Animal {
  ....
  boolean endangers(Animal a) {
    return (a.slowerThan(100)
            && a.isLighter(this.weight/2));
  }
}

class Rattlesnake extends Snake {
   ...
   boolean endangers(Animal a) {
     return (!a.hasThickSkin()
             || super.endangers(a))
   }
}

Animal a = new Rattlesnake(...);
Animal b = new Mouse(...);

a.endangers(b);
```

dynamic

static

Part 2

Class Language with Explicit Static Calls

```
<Class>   ::= class <Symbol>(<Field>, ...):   NEW
                <Method>
                ...
<Field>   ::= <Symbol>
<Method>  ::= method <Symbol>(arg): <Exp>
```

```
<Exp> ::= <Int>
        | <Exp> + <Exp>
        | <Exp> * <Exp>
        | arg
        | this
        | new <Symbol>(<Exp>, ...)                NEW
        | <Exp>.<Symbol>
        | <Exp>.<Symbol>(<Exp>)
        | (<Exp> :: <Symbol>).<Symbol>(<Exp>)     NEW
```

```
class Posn(x, y):
  method mdist(arg):
    this.x + this.y
  method addDist(arg):
    arg.mdist(0) + this.mdist(0)
new Posn(1, 2).addDist(new Posn(3, 4))
```

Analogous Java code

```
class Posn {
  int x, y;
  int mdist() {
    return this.x + this.y;
  }
  int addDist(Posn p) {
    return p.mdist() + mdist();
  }
}
new Posn(1,2).mdist(new Posn(3,4))
```

Class Language with Explicit Static Calls

```
<Class>   ::= class <Symbol>(<Field>, ...):          NEW
                <Method>
                ...
<Field>   ::= <Symbol>
<Method>  ::= method <Symbol>(arg): <Exp>
```

```
<Exp>  ::= <Int>
         | <Exp> + <Exp>
         | <Exp> * <Exp>
         | arg
         | this
         | new <Symbol>(<Exp>, ...)                 NEW
         | <Exp>.<Symbol>
         | <Exp>.<Symbol>(<Exp>)
         | (<Exp> :: <Symbol>).<Symbol>(<Exp>)      NEW
```

```
class Posn(....):
  method addDist(arg):
    arg.mdist(0) + this.mdist(0)
class Posn3D(x, y, z):
  method mdist(arg):
    this.z + (this :: Posn).mdist(arg)
  method addDist(arg):
    (this :: Posn).addDist(arg)
new Posn3D(1, 2, 3).addDist(new Posn(3, 4))
```

Analogous Java code

```
class Posn {
  .... as before ....
}
class Posn3D extends Posn {
  int z; ....
  int mdist() {
    return this.z + super.mdist();
  }
  int addDist(Posn p) {
    return super.addDist(p);
  }
}
new Posn3D(1,2,3).addDist(new Posn(3,4))
```

Part 3

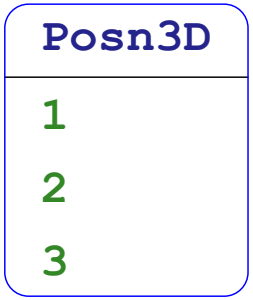
Object Values

How does

```
new Posn3D(1, 2, 3).mdist(....)
```

dispatch to the right `mdist`?

The result of `new Posn3D(1, 2, 3)` can hold a class tag and field values:



Look for field names and methods in the class

Classes and Object Values

```
type Value
| intV(n :: Int)
| objV(class_name :: Symbol,
       field_values :: Listof(Value))

type Class
| classC(field_names :: Listof(Symbol),
          methods :: Listof(Symbol * Exp))
```

```
interp :: (Exp, Listof(Symbol * Class), Value, Value) -> Value
```

Examples

```
check: interp(intE(10),
              [],
              objV(#'Object, []),
              intV(0))
       ~is intV(10)
```

Examples

```
def posn_class:
  values(#'Posn,
         classC([#'x, #'y],
                [values(#'mdist,
                        plusE(getE(thisE(), #'x), getE(thisE(), #'y))),
                 values(#'addDist,
                        plusE(sendE(thisE(), #'mdist, intE(0)),
                              sendE(argE(), #'mdist, intE(0))))]))
def posn3D_class:
  values(#'Posn3D,
         classC([#'x, #'y, #'z],
                [values(#'mdist,
                        plusE(getE(thisE(), #'z),
                              ssendE(thisE(), #'Posn, #'mdist, argE()))),
                 values(#'addDist,
                        ssendE(thisE(), #'Posn, #'addDist, argE()))]))

fun interp_posn(a):
  interp(a, [Posn_class, Posn3D_class], objV(#'Object, []), intV(0))
```

Examples

```
check: interp_posn(newE(#'Posn,
                        [intE(2),
                         intE(7)]))
       ~is objV(#'Posn,
                [intV(2),
                 intV(7)])
```

```
def new_posn27:
  newE(#'Posn, [intE(2), intE(7)])
```

Examples

```
def posn_class:
  values(#'Posn,
         classC([#'x, #'y],
                [values(#'mdist,
                        plusE(getE(thisE(), #'x), getE(thisE(), #'y))),
                 values(#'addDist,
                        plusE(sendE(thisE(), #'mdist, intE(0)),
                              sendE(argE(), #'mdist, intE(0))))]))

def new_posn27:
  newE(#'Posn, [intE(2), intE(7)])
```

```
check: interp_posn(sendE(new_posn27, #'mdist, intE(0)))
       ~is intV(9)
```

Examples

```
def posn_class:
  values(#'Posn,
         classC([#'x, #'y],
                [values(#'mdist,
                        plusE(getE(thisE(), #'x), getE(thisE(), #'y))),
                 values(#'addDist,
                        plusE(sendE(thisE(), #'mdist, intE(0)),
                              sendE(argE(), #'mdist, intE(0))))]))
def posn3D_class:
  values(#'Posn3D,
         classC([#'x, #'y, #'z],
                [values(#'mdist,
                        plusE(getE(thisE(), #'z),
                              ssendE(thisE(), #'Posn, #'mdist, argE()))),
                 values(#'addDist,
                        ssendE(thisE(), #'Posn, #'addDist, argE()))]))
def new_posn27:
  newE(#'Posn, [intE(2), intE(7)])
def new_posn531:
  newE(#'Posn3D, [intE(5), intE(3), intE(1)])
```

```
check: interp_posn(sendE(new_posn531, #'addDist, new_posn27))
       ~is intV(18)
```

Part 4

Interpreter

```
def interp :: (Exp, Listof(Symbol * Class), Value, Value) -> Value:
  fun (a, classes, this_val, arg_val):
    fun recur(expr):
      interp(expr, classes, this_val, arg_val)
    match a
    | ....
    | intE(n): intV(n)
    | plusE(l, r): num_plus(recur(l), recur(r))
    | multE(l, r): num_mult(recur(l), recur(r))
    | thisE(): this_val
    | argE(): arg_val
    | ....
```

Interpreter

```
def interp :: (Exp, Listof(Symbol * Class), Value, Value) -> Value:
  fun (a, classes, this_val, arg_val):
    fun recur(expr):
      interp(expr, classes, this_val, arg_val)
    match a
    | ....
    | newE(class_name, field_exprs):
        def c = find(classes, class_name)
        def vals = map(recur, field_exprs)
        if length(vals) == length(classC.field_names(c))
        | objV(class_name, vals)
        | error(#'interp, "wrong field count")
    | ....
```

Interpreter

```
def interp :: (Exp, Listof(Symbol * Class), Value, Value) -> Value:
  fun (a, classes, this_val, arg_val):
    fun recur(expr):
      interp(expr, classes, this_val, arg_val)
    match a
    | ....
    | getE(obj_expr, field_name):
        match recur(obj_expr)
        | objV(class_name, field_vals):
            match find(classes, class_name)
            | classC(field_names, methods):
                find(map2(fun (n, v): values(n, v),
                          field_names,
                          field_vals),
                     field_name)
        | ~else: error(#'interp, "not an object")
    | ....
```

Interpreter

```
def interp :: (Exp, Listof(Symbol * Class), Value, Value) -> Value:
  fun (a, classes, this_val, arg_val):
    fun recur(expr):
      interp(expr, classes, this_val, arg_val)
    match a
    | ....
    | sendE(obj_expr, method_name, arg_expr):
        def obj = recur(obj_expr)
        def arg_val = recur(arg_expr)
        match obj
        | objV(class_name, field_vals):
            call_method(class_name, method_name, classes,
                        obj, arg_val)
        | ~else: error(#'interp, "not an object")
    | ....
```

Calling a Method

```
fun call_method(class_name, method_name, classes,
                obj, arg_val):
  match find(classes, class_name)
  | classC(field_names, methods):
      let body_expr = find(methods, method_name):
        interp(body_expr,
               classes,
               obj,
               arg_val)
```

Interpreter

```
def interp :: (Exp, Listof(Symbol * Class), Value, Value) -> Value:
  fun (a, classes, this_val, arg_val):
    fun recur(expr):
      interp(expr, classes, this_val, arg_val)
    match a
    | ....
    | ssendE(obj_expr, class_name, method_name, arg_expr):
        def obj = recur(obj_expr)
        def arg_val = recur(arg_expr)
        call_method(class_name, method_name, classes,
                    obj, arg_val)
    | ....
```

Part 5

Subclasses

Subclasses with **`Exp`**:

```
class Posn(x, y):
  method mdist(arg):
    this.x + this.y
  method addDist(arg):
    arg.mdist(0) + this.mdist(0)

class Posn3D(x, y, z):
  method mdist(arg):
    this.z + (this :: Posn).mdist(arg)
  method addDist(arg):
    arg.mdist(0) + this.mdist(0)

new Posn3D(1, 2, 3).addDist(new Posn(3, 4))
```

Programmer manually

- duplicates fields
- implements method inheritance

Subclasses

`ExpI` adds ***implementation inheritance***:

```
class Posn(x, y):
  extends Object
  method mdist(arg):
    this.x + this.y
  method addDist(arg):
    arg.mdist(0) + this.mdist(0)

class Posn3D(z):
  extends Posn
  method mdist(arg):
    this.z + super.mdist(arg)

new Posn3D(1, 2, 3).addDist(new Posn(3, 4))
```

Class Language with Inheritance

```
<Class>   ::=  class <Symbol>(<Field>, ...):          NEW
                  extends <Symbol>
                  <Method>
                  ...
<Field>   ::=  <Symbol>
<Method>  ::=  <Symbol>(arg(), <Exp>)

<Exp>  ::=  <Int>
         |  <Exp> + <Exp>
         |  <Exp> * <Exp>
         |  arg
         |  this
         |  new <Symbol>(<Exp>, ...)
         |  <Exp>.<Symbol>
         |  <Exp>.<Symbol>(<Exp>)
         |  super.<Symbol>(<Exp>)                   NEW
```

Compiling Inheritance

```
class Posn(x, y):
  extends Object
  method mdist(arg):
    this.x + this.y
  method addDist(arg):
    arg.mdist(0) + this.mdist(0)

class Posn3D(z):
  extends Posn
  method mdist(arg):
    this.z + super.mdist(arg)

new Posn3D(1, 2, 3).addDist(new Posn(3, 4))
```

```
class Posn(x, y):
  method mdist(arg):
    this.x + this.y
  method addDist(arg):
    arg.mdist(0) + this.mdist(0)

class Posn3D(x, y, z):
  method mdist(arg):
    this.z + (this :: Posn).mdist(arg)
  method addDist(arg):
    arg.mdist(0) + this.mdist(0)

new Posn3D(1, 2, 3).addDist(new Posn(3, 4))
```

- merge fields from superclasses
- change **`super`** to **`(this :: <Symbol>)`**
- merge/override methods

Part 6

Classes

```
type ClassI
| classI(super_name :: Symbol,
          field_names :: Listof(Symbol),
          methods :: Listof(Symbol * ExpI))
```

Expressions

```
type ExpI
| intI(n :: Int)
| plusI(lhs :: ExpI,
        rhs :: ExpI)
| multI(lhs :: ExpI,
        rhs :: ExpI)
| argI()
| thisI()
| newI(class_name :: Symbol,
       args :: Listof(ExpI))
| getI(obj_expr :: ExpI,
       field_name :: Symbol)
| sendI(obj_expr :: ExpI,
        method_name :: Symbol,
        arg_expr :: ExpI)
| superI(method_name :: Symbol,
         arg_expr :: ExpI)
```

Examples

```
check: exp_i_to_c(intI(10))
       ~is intE(10)
```

Examples

```
check: exp_i_to_c(thisI())
       ~is thisE()
```

Examples

```
check: exp_i_to_c(superI(#'mdist, intI(0)))
       ~is ssendE(thisE(), ???, #'mdist, intE(0))
```

Examples

```
exp_i_to_c :: (ExpI, Symbol) -> Exp

check: exp_i_to_c(superI(#'mdist, intI(0)), #'Posn)
       ~is ssendE(thisE(), #'Posn, #'mdist, intE(0))
```

Compiling Expressions

```
fun exp_i_to_c(a :: ExpI, super_name :: Symbol) :: Exp:
  fun recur(expr):
    exp_i_to_c(expr, super_name)
  match a
  | intI(n): intE(n)
  | plusI(l, r): plusE(recur(l), recur(r))
  | multI(l, r): multE(recur(l), recur(r))
  | ....
  | superI(method_name, arg_expr):
      ssendE(thisE(),
             super_name,
             method_name,
             recur(arg_expr))
```

Compiling Class Methods

```
fun class_i_to_c_not_flat(c :: ClassI) :: Class:
  match c
  | classI(super_name, field_names, methods):
      classC(field_names,
             map(fun (m):
                   values(fst(m),
                          exp_i_to_c(snd(m), super_name)),
                 methods))
```

Flattening a Class

```
fun flatten_class(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  ....
```

Flattening a Class

```
fun flatten_class(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  ....  find(classes_not_flat, name) ....
```

Flattening a Class

```
fun flatten_class(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  match find(classes_not_flat, name)
  | classC(field_names, methods):
      ....
```

Flattening a Class

```
fun flatten_class(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  match find(classes_not_flat, name)
  | classC(field_names, methods):
      .... flatten_super(name, classes_not_flat, i_classes) ....
```

Flattening a Class

```
fun flatten_class(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  match find(classes_not_flat, name)
  | classC(field_names, methods):
      match flatten_super(name, classes_not_flat, i_classes)
      | classC(super_field_names, super_methods):
          ....
```

Flattening a Class

```
fun flatten_class(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  match find(classes_not_flat, name)
  | classC(field_names, methods):
      match flatten_super(name, classes_not_flat, i_classes)
      | classC(super_field_names, super_methods):
          classC(....,
                 ....)
```

Flattening a Class

```
fun flatten_class(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  match find(classes_not_flat, name)
  | classC(field_names, methods):
      match flatten_super(name, classes_not_flat, i_classes)
      | classC(super_field_names, super_methods):
          classC(add_fields(super_field_names,
                            field_names),
                 ....)
```

Flattening a Class

```
fun flatten_class(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  match find(classes_not_flat, name)
  | classC(field_names, methods):
      match flatten_super(name, classes_not_flat, i_classes)
      | classC(super_field_names, super_methods):
          classC(add_fields(super_field_names,
                            field_names),
                 add_or_replace_methods(super_methods,
                                        methods))
```

Flattening a Class

```
fun flatten_super(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  ...
```

Flattening a Class

```
fun flatten_super(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  ... find(i_classes, name) ...
```

Flattening a Class

```
fun flatten_super(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  match find(i_classes, name)
  | classI(super_name, field_names, i_methods):
      ...
```

Flattening a Class

```
fun flatten_super(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  match find(i_classes, name)
  | classI(super_name, field_names, i_methods):
      ... flatten_class(super_name,
                        classes_not_flat,
                        i_classes)
      ...
```

Flattening a Class

```
fun flatten_super(name :: Symbol,
                  classes_not_flat :: Listof(Symbol * Class),
                  i_classes :: Listof(Symbol * ClassI)) :: Class:
  match find(i_classes, name)
  | classI(super_name, field_names, i_methods):
      if super_name == #'Object
      | classC([], [])
      | flatten_class(super_name,
                      classes_not_flat,
                      i_classes)
```

Interpreter

```
fun interp_i(i_a :: ExpI,
             i_classes :: Listof(Symbol * ClassI)) :: Value:
  def a = exp_i_to_c(i_a, #'Object)
  def classes_not_flat:
    map(fun (i):
          values(fst(i),
                 class_i_to_c_not_flat(snd(i))),
        i_classes)
  def classes:
    map(fun (c):
          let name = fst(c):
            values(name,
                   flatten_class(name, classes_not_flat, i_classes)),
        classes_not_flat)
  interp(a, classes, objV(#'Object, []), intV(0))
```