9/9. CS339

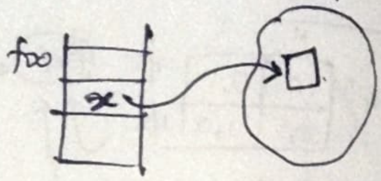
Java's Object Model.

```
foo() {
  A x;
    ↳ Just Reserves space (in stack).
      for A type variable
      during compilation.
}
```

x = new A();
↳ Constructor.

↓

Object allocation.

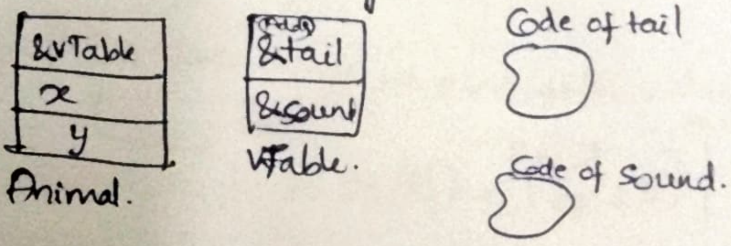


→ Values of fields are stored in heap. ~~That points x~~ points to that location.

→ Every non-primitive variable is a pointer.

For eg. int x → Primitive

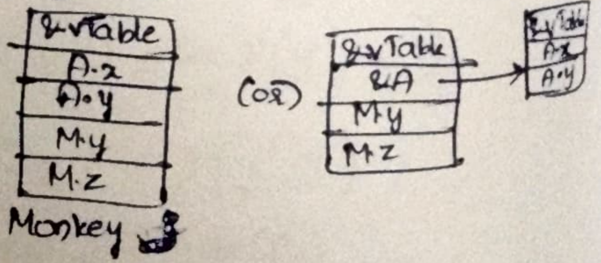
→ Def^n of methods doesn't change at runtime. So, the code of method can be stored at some memory.



If Objects were allowed to change code at runtime, then, this way of storing code is useless.

→ Fields values will be diff for each object. (of x & y)

* Fields are extended to child class.



* Animal W = new A();
or
new M();

but

Monkey W = new A(); → Not allowed. → We get error.

→ Static Vs. Dynamic
(Compile Time) (Run Time).

foo (int, bool)
foo (int, int).

If foo (5, true);
↓
Can it statically make a decision about what fn is called?
↓ ↓
Yes. Decide. (Based on arguments)

→ Overloading | Static | Compile Time Polymorphism.

A x;
x.foo();

→ If arguments are same, but it is defined differently in 2 classes, in which 1 is subclass to other, then;

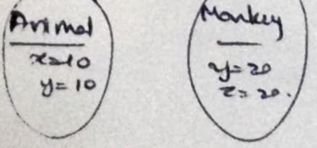
↓
Run Time Polymorphism. / Overriding.

→ If we don't find the variable in child object, it looks for parent object. → Inheritance

→ P2's type is Animal, we don't know whether it stores animal / monkey object type. So, if fields are to be bound statically, then P2.y will be 10 but not 20.

* Run type polymorphism → Exists for methods but not for fields.

→ In Java, Fields are bound / resolved statically.



Animal P2;
P2 = new Monkey();
Print (P2.x   P2.y   ((Monkey)P2).y   (Monkey)P2).z);
↓ 20   ↓ 10   ↓ 20   ↓ 20.

→ Java supports Run Time poly.

P2.sound(); ⟶ Animal's sound (As monkey Object has no sound() method).

P2.tail(); ⟶ Monkey's Tail

((Animal) p2).tail(); ⟶ Monkey's Tail

→ Object is making decision. (Objects Dispatches msg.)
P2 is pointing to Monkey object.

→ Field , Method
↓ ↓
Static time Run time.

So, casting types will ~~the~~ work for fields,
but not for methods.

```
(lambda (msg)
  (cond ((eq? msg 'tail) (tail))
        ((eq? msg 'sound) (super 'sound)).
let ((super (Animal))
```