This is for the loyal Schemers and MLers.

```
interface T^𝓘 {
  o→o^𝓘 apply(T^𝓘 x);
}
```

```
interface o→o^𝓘 {
  Object apply(Object x);
}
```

```
interface oo→oo^𝓘 {
  o→o^𝓘 apply(o→o^𝓘 x);
}
```

```
interface oo→oo→oo^𝓘 {
  o→o^𝓘 apply(oo→oo^𝓘 x);
}
```

```
class Y implements oo→oo→oo^𝓘 {
  public o→o^𝓘 apply(oo→oo^𝓘 f) {
    return new H(f).apply(new H(f)); }
}
```

```
class H implements T^𝓘 {
  oo→oo^𝓘 f;
  H(oo→oo^𝓘 _f) {
    f = _f; }
  public o→o^𝓘 apply(T^𝓘 x) {
    return f.apply(new G(x)); }
}
```

```
class G implements o→o^𝓘 {
  T^𝓘 x;
  G(T^𝓘 _x) {
    x = _x; }
  public Object apply(Object y) {
    return (x.apply(x)).apply(y); }
}
```

No, we wouldn't forget factorial.

```
class MkFact implements oo→oo^𝓘 {
  public o→o^𝓘 apply(o→o^𝓘 fact) {
    return new Fact(fact); }
}
```

```
class Fact implements o→o^𝓘 {
  o→o^𝓘 fact;
  Fact(o→o^𝓘 _fact) {
    fact = _fact; }
  public Object apply(Object i) {
    int inti = ((Integer)i).intValue();
    if (inti == 0)
      return new Integer(1);
    else
      return
        new Integer(
          inti
          *
          ((Integer)
           fact.apply(new Integer(inti − 1)))
          .intValue()); }
}
```