6-1

```
public static void main (String [] args) {
    return main (args);
}
```

main(args, append(args))

⇒ Stack Overflow

at C1.main(t.java:3) × (100 to 1033)

CEK ⟶ continuation
↳ code ⟶ env
env ↓ Heap
continuation ↓ stack

$\Omega = (\lambda x.\, x\,x)^{\leftarrow w}\,(\lambda x.\, x\,x)^{\leftarrow w}$ $\quad \Omega \to_\beta \Omega$

$\langle (\lambda f.\,(\lambda x.\, f\,(x\,x))\,(\lambda x.\, f\,(x\,x)))\,(\lambda y.\, y),\ \emptyset^{=E_0},\ mt\rangle$

$\to^3$ (a) $\langle (\lambda x.\, f(x\,x))\,(\lambda x.\, f(x\,x)),\ [f \mapsto clo(\lambda y.\, y, \emptyset)]^{=E_1},\ mt\rangle$

$\langle (\lambda x.\, f(x\,x))^{=wf},\ E_1,\ fun((\lambda x.\, f(x\,x)), E_1, mt)\rangle$

$\langle clo(wf, E_1), E_1, fun(wf, E_1, mt)\rangle$

$\langle wf, E_1, arg(clo(wf, E_1), mt)\rangle$

$\langle clo(wf, E_1), E_1, arg(clo(wf, E_1), mt)\rangle$

$\langle f\,(x\,x),\ E_1[x \mapsto clo(wf, E_1)],\ mt\rangle$

$\to$ (a') $\langle (x\,x),\ E_1[x \mapsto clo(wf, E_1)],\ arg(clo(\lambda y.\, y, \emptyset), mt)\rangle$

$\langle \Omega, \emptyset, mt\rangle$

$\langle w, \emptyset, fun(w, \emptyset, mt)\rangle$

$\langle clo(w, \emptyset), \emptyset, \text{''}\rangle$

$\langle w, \emptyset, arg(clo(w, \emptyset), mt)\rangle$

(A) $\langle x\,x, \emptyset[x \mapsto clo(w, \emptyset)]^{=E_1}, mt\rangle$

$\langle x, E_1, fun(x, E_1, mt)\rangle$

$\langle x, E_1, arg(clo(w, \emptyset), mt)\rangle$

(A) $\langle x, \emptyset[x \mapsto clo(w, \emptyset)], mt\rangle$

with constant heap + stack space

$\langle V, E, arg(clo(x, e, E'), k) \rangle$

CEK rule $\mapsto \langle e, E'[x \mapsto V], k \rangle$

"A CEK funcall reduces stack space"

Java Rule $\mapsto \langle e, E'[x \mapsto v], same(k) \rangle$

new rule $\langle V, E, same(k) \rangle \mapsto \langle V, E, k \rangle$

$\langle w, \emptyset, arg(clo(w, \emptyset), mt) \rangle$
$\langle clo(w, \emptyset), arg(clo(w, \emptyset), mt) \rangle$
(A) $\langle x\ x, \emptyset[x \mapsto clo(w, \emptyset)]^{=E_1}, same(mt) \rangle$
$\langle x, E_1, fun(x, E_1, same(mt)) \rangle$
$\langle x, E_1, arg(clo(w, \emptyset), same(mt)) \rangle$
(A') $\langle x\ x, E_1, same(same(mt)) \rangle$

Java is { "not safe-for-space"
"do not have proper tail-calls"

CEK { "tail-call optimizing"
The stack is for evaluate function arguments
NOT calling functions
(A function is a parameterized GOTO)

```
int f (x) {
  return 1 + f(x)
}
```

"Java" puts in same(k)

- Pascal
- Python
- Ruby
- JS

looks at k

Java puts in funcall(T, k)
where T is the "trust" or not flag

CEK CETK

<stoptrusting e, E, T, k>
↦ <e, E, False, trustme(k)>
<V, E, T, trustme(k)>
↦ <V, E, True, k>
<inspect, E, T, k> looks at k
↦ <T, E, T, k>

Stack Inspection

---

k = mt
| fun(M, E, k)
| arg(V, k)
| untrusted(k) ← new

Φ(mt) = Y
Φ(fun(m, e, k)) = Φ(k)
Φ(arg(V, k)) = Φ(k)
Φ(un(k)) = N

<stoptrusting e, E, k>
↦ <e, E, U(k)>
<V, E, untrusted(k)>
↦ <V, E, k>
<inspect, E, k>
↦ <Φ(k), E, k>

U : k → k
U(mt) = un(mt)
U(fun ...) = un(fun ...)
U(arg ...) = un(arg ...)
U(untrusted(k)) = untrusted(k)

John Clements

```
cd src ;;
make ;;
cd ...
```

```
cd src ;; make
cd ..
```

```
if [-d src]
  cd src ;; (
    make;
    cd ... )
```

```
e = ...
  | with-cwd( d, e)
  | read-cwd()
```

```
k = ...
  | cwd(d, k)
```

Generalize...

```
e = ...
  | with-continuation-mark(k, v, e)
  | read-continuation-marks(k)
k = ...
  | wcm(k, v, k)
```

$u(v_k, v_v, k)$

$\equiv u(v_k, v_v, wcm(v_k, v_v, k))$

$= wcm(v_k, v_v, k)$