

Exercise 10.3 Extend the micro-Java interpreter in `oo/oo.sml` to implement static fields as introduced in Exercise 10.2 above. You need to add a static field environment `sfields` to the runtime representation of every class, that is, the SML type `class`. A static field environment maps a field name to a store location (and then the store maps the location to the current value of the field). All objects of the class have the same static field environment.

You will need to modify the `class` type and the functions `initialize`, `eval`, and `exec`. In the `initialize` function, you must make the `fieldsmethods` function return three lists for a class: a list of the static fields, a list of the non-static fields, and a list of the methods. The list of static fields should be turned into a static field environment `sfields : location env`, and locations should be allocated in the store for the static fields. Thus it may be necessary to pass the store `sto0` to the `initialize` function.

Here's a complication you can ignore: In Java, if a class `A` declares a static field `g`, and `B` is a subclass of `A`, then `A`'s static field `g` is accessible also as `B.g` (unless `B` declares another `g` field, or `A`'s `g` field is private):

```
class A {
  static int f = 1;
  static int g = 2;
}

class B extends A {
  static int f = 3;
}

class Superclassfields {
  public static void main(String[] args) {
    System.out.println("A.f = " + A.f);
    System.out.println("A.g = " + A.g);
    System.out.println("B.f = " + B.f);
    System.out.println("B.g = " + B.g);
  }
}
```

In Java this program prints

```
A.f = 1
A.g = 2
B.f = 3
B.g = 2
```

In micro-Java you can just assume that the correct class prefix is always given in a field access `A.g` or `A.g = ...`

Exercise 10.4 Add static methods to micro-Java (first to the abstract syntax and interpreter, then to the lexer and parser specifications). This can be done by adding a static method environment to every class. A static method is called just as a non-static one, but the current object reference `this` is not passed as a parameter to the method. Hence there is no way a static method can access non-static fields or non-static methods.

Assume that static method calls are prefixed with the correct class name `C`, using C++ syntax, as in `C::m(...)`.

Exercise 10.5 Permit a `super` reference in non-static method calls, as in `super.m(...)`. If the current object reference `this` refers to an object `o` of class `C`, then the search for method `m` starts in the immediate superclass of `C`, not in class `C`. The method is called with its current object reference `this` bound to `o`.

Exercise 10.6 Implement non-static field access `o.f` in micro-Java properly (as in Java). Modify the abstract syntax for non-static field access (`GetField` and `SetField`) to carry also the compile-time type (a class name) of the object expression `o`. Modify the field environment of objects so that it maps from a pair (B, f) of (declaring) class `B` and field name `f` to a location, not just from the field name `f` to a location. Modify non-static field access to take the type of the object expression `o` into account as in Java.

Exercise 10.7 (Small project) Write a type checker for micro-Java. For this, you need to modify the abstract syntax to carry a type (primitive type or class) on every field access (`GetField` and `SetField`).