CSE 505, Fall 2005, Assignment 5
Due: Friday 9 December 2005, 4:30PM

hw5.tar, available on the course website, contains a Caml file and a Java file you will need.

Last updated: November 27

Introduction: We investigate programming in an OO-style in a functional language and programming in a
functional-style in an OO language. Doing so reveals connections between functions and objects and shows
that one language’s semantics can be approximated (with some pain) via a programming style in another
language. Do not use Caml’s OO features or fancy Java features (anonymous inner classes and generics).

1. hw5.ml defines the type obj as either a floating-point number\(^1\) or a list of fields (a mapping from
strings to references-to-objects) and a list of methods (a mapping from strings to functions). Fields
are mutable. A function used to define a method expects “self” (a.k.a. “this”) as its first argument
and the method arguments in a list. A constructor is just an ML function that returns a new object.
To simulate subclassing, one constructor can call another. To make this easier, if a field or method list
has the same string multiple times, the one nearest the beginning of the list overrides any others.

You are provided functions for extracting an object’s fields and methods (if it is not a number) or the
underlying float (if it is a number). You are also provided a constructor for 2D points, but it uses some
functions you must implement.

(a) Implement get, an ML function that takes a string s and object o and returns the contents of o’s
s field (raising an ObjectExpected or FieldNotFound exception as appropriate).

(b) Implement set, an ML function that takes a string s, object o, and object v and changes o’s s
field to hold v (raising an ObjectExpected or FieldNotFound exception as appropriate).

(c) If you haven’t already, have get and set share a helper function so get and set occupy 1 line.
(Hint: Have the function take a function of type obj ref -> 'a and other arguments.)

(d) Implement findAndSend, an ML function that takes a string s, object rcvr (pronounced “re-
ceiver”), method list ms, and arguments args. It should find the method “named” s in ms
(raising MethodNotFound if none exists) and return the result of calling that method with rcvr
and args.

(e) Implement send using findAndSend, where send takes a string s, object rcvr, and argument list
args and returns the result of sending s to rcvr with args.

(f) Implement getter, which takes a string s and returns a function that can be used for a method
that returns the contents of the field s.

(g) Implement setter, which takes a string s and returns a function that can be used for a method
that sets the contents of the field s and then returns the receiver. Hint: You can use List.hd to
get the first argument.
At this point, newPoint should work.

(h) Change newPoint so the object it produces also has a method distToOrigin2, which is just like
distToOrigin1 except it uses the getX and getY methods instead of directly reading fields.

(i) Write a constructor newPoint3D for 3D points. Use newPoint as a helper function (i.e., a super-
class).

- Add a field z and methods getZ and setZ with the obvious behavior.
- Override distToOrigin1 and distToOrigin2 so they properly account for the z-dimension.
  As with 2D points, have distToOrigin1 read fields directly and distToOrigin2 call getter
  functions.
- Add a method distToOrigin3 that uses the distToOrigin1 defined in newPoint as a helper
  method (i.e., get the result of this “super” call, square it, add the square of the z-field, and
take the square-root). Hint: Use findAndSend.

---

\(^1\)A more OO approach would treat numbers as objects. For this homework, such objects would need many methods
(multiplication, addition, trigonometry, conversion to strings), so we are not going to bother.
(j) Write a constructor `newPolarPoint` for 2D points. Use `newPoint` as a helper function (i.e., a superclass).

- Although objects created by `newPolarPoint` will have fields x and y (initialized by the call to `newPoint`), do not use them. Instead, add fields r and theta (computed by calling `toPolar`).
- Override `getX`, `getY`, `setX`, and `setY` to use r and theta. The simplest implementations use `toCartesian` (and for the setters also `toPolar`).
- In a comment in the code, explain why sending `distToOrigin1` and `distToOrigin2` to a polar-point gives different results.

(k) Write a constructor `newPolarPointB` for a “subclass” of `newPolarPoint`.

- Override `distToOrigin1` so that it’s simply a getter for the r field.
- In a comment in the code, explain why sending `distToOrigin1` and `distToOrigin2` to such a point gives the same results.

2. `hw5.java` defines a class `MLList` for ML-style lists where list elements have type `Object` and we use `null` for the empty-list. You will implement 9 static methods. Implement them as an ML programmer naturally would. For example, do not use explicit loops.

(a) Implement the static methods `cons`, `length`, `append`, `filter`, and `fold_left`. Their behavior should be analogous to the functions of the same name in the Caml library. Because list fields are `final`, you need not worry about sharing. Note you will have to define `BoolClosure` and `BinaryClosure` appropriately.

(b) Implement the static methods in class `C` as described below. For each, you will need also to define a class that implements an appropriate interface. Two of the methods make use of the `Integer` class defined in `java.lang` (because `int` is not a subtype of `Object` but `Integer` is).

- `noNulls` returns a list that is like its input with all `null` entries removed. Use `filter`.
- `numsLessThan` returns a list that is like its first input with everything removed that is not a non-null `Integer` with a value strictly less than its second input. Use `filter`.
- `numNotNull` returns an `int` that is the number of non-null entries in its input. Use `fold_left`. (Hint: Use `Integer` where you want an `int` but `fold_left` demands an `Object`.)
- `sumNumsLessThan` returns an `int` that is the sum of the entries in its first input that are non-null `Integer` objects with value strictly less than its second input. Use `fold_left`. (Hint: Use `Integer` where you want an `int` but `fold_left` demands an `Object`.)

3. (Extra Credit) Make all these improvements to the code for problem 1. Use a separate copy of `hw5.ml`.

- Change the constructors to use less space for method lists. In particular, all objects created by a constructor should share a method list rather than allocating a new one for each object. Do not change the definition of `obj` for this purpose.
- Change the definition of `meth` so that methods have a fixed arity (number of arguments). Change message-send so that it invokes a method only if the number of arguments matches. In other words, support static overloading, but since every object has the same type, the only possible overloading is arity.
- Support “instance-of”: Implement a way to test dynamically whether an object was created by a constructor or one of its super-class constructors. You may change the definition of `obj` for this purpose.

What to turn in:

- Caml source code in `hw5.ml` and Java source code in `hw5.java`.
- Put the extra credit in a file named `hw5_extra.ml`.

Email your code to Erika as `firstname-lastname--hw5.tgz` or `firstname-lastname--hw5.zip`.