CS 5510 Homework 8

Due: Friday, October 30th, 2015 11:59pm

Start with `class-inherit.rkt`, which combines `class.rkt`, `inherit.rkt`, and `inherit-parse.rkt` into a single file.

You may find it nicer to work with `class.rkt`, `inherit.rkt`, and `inherit-parse` as separate files, but you must combine them into a single file for handin. To combine the files: append them in the listed order, then remove the second and third `#lang plai-typed` lines, remove all `require` forms, and add back just `(require plai-typed/s-exp-match)`.

Part 1 — Instiantiating `object`

In the starting code, `{new object}` doen't work, even though `object` is supposed to be a built-in class with no fields and no methods. Fix the implementation to that `{new object}` produces an instance of `object`.

```
(test (interp-prog (list)
                   '{new object})
      `object)
```

Part 2 — Conditional via `select`

Add `select`:

```
<Expr> = ...
       | {select <Expr> <Expr>}
```

The first expression in `select` should produce a number result, while the second expression should produce an object. If the number is 0, `select` calls the `zero` method of the object. If the number is not 0, `select` calls the `nonzero` method of the object. In each case, `select` calls the method with the argument `0`.

```
(test (interp-prog (list '{class snowball extends object
                            {size}
                            {zero this}
                            {nonzero {new snowball {+ 1 {get this size}}}}})
                   '{get {select 0 {new snowball 1}} size})
      '1)
(test (interp-prog (list '{class snowball extends object
                            {size}
                            {zero this}
                            {nonzero {new snowball {+ 1 {get this size}}}}})
                   '{get {select {+ 1 2} {new snowball 1}} size})
      '2)
```

The result is up to you when `select` gets a non-number result for its first subexpression, but a "not a number" error is sensible. Similarly, the result is up to you when the second subexpression's result is not an object or does not have a `zero` or `nonzero` method.

You will probably find it easiest to extend and test the `ExprC` layer, then then `ExprI` layer, and then the `parse` and `interp-prog` layer.

Part 3 — `instanceof`

Add `instanceof`:

```
<Expr> = ...
       | {instanceof <Expr> <Sym>}
```

An expression `{instanceof <Expr> <Sym>}` produces 0 if the result of `<Expr>` is an object that is an instance of the class named by `<Sym>` or any of its subclasses. It should produce 1 f the result of `<Expr>` is any other object. Note that `{instanceof <Expr> object}` should produce 0 as long as the value of `<Expr>` is an object.

Note that you will have to introduce a notion of superclasses into the `ClassC` layer, even though method inheritance remains the job of the `ClassI` layer.

Example:

```
(test (interp-prog (list '{class fish extends object
                           {size color}})
                   '{instanceof {new fish 1 2} fish})
      '0)

(test (interp-prog (list '{class fish extends object
                           {size color}})
                   '{instanceof {new fish 1 2} fish})
      '0)
(test (interp-prog (list '{class fish extends object
                           {size color}}
                         '{class shark extends fish
                           {teeth}})
                   '{instanceof {new shark 1 2 3} fish})
      '0)
```

The result is up to you if `<Expr>` in `{instanceof <Expr> <Sym>}` does not produce an object or if `<Sym>` is not the name of a class. If you continue to part 4, however, all values count as objects.

Part 4 — Extra Credit: Numbers as Objects

Our language currently has objects and numbers. We could claim more conceptual purity if we say that numbers are also objects, number constants are just a shorthand for instantiating number objects, and the `+`, `*`, and `select` forms are just shorthands for calling `plus`, `mult`, and `select` methods of number objects.

Change the language so that number values are treated as objects and method calls to `plus`, `mult`, and `select` work on numbers. No other methods should work on numbers, and no fields should be available from numbers.

Although numbers count as objects, `interp-prog` should continue to treat numbers specially, returning a number S-expression instead of `` `object`` to reflect a number value.

Examples:

```
(test (interp-prog (list)
                   '{instanceof 8 object})
      '0)
(test (interp-prog (list)
                   '{send 8 plus 9})
      '17)
(test (interp-prog (list)
                   '{send 8 mult 9})
      '72)
(test (interp-prog (list '{class snowball extends object
                           {size}
                           {zero this}
                           {nonzero {new snowball {+ 1 {get this size}}}}})
                   '{get {send 8 select {new snowball 10}} size})
      '11)
```

The result of an expression `{send 8 plus {new object}}` or `{+ 8 {new object}}` is up to you, but a “not a number” error is sensible. For `{send {new object} plus 8}`, the error surely must be “not found”, since the object has no `plus` method. For `{+ {new object} 8}`, either “not a number” or “not found” could be sensible, but only “not found” is sensible if you implement part 5.

Part 5 — More Extra Credit: Objects as Numbers

In addition to treating numbers as objects, allow objects to be used as numbers, so that `+`, `*`, and `select` are equivalent to method calls: `{+ e1 e2}` is the same as `{send e1 plus e2}`, `{* e1 e2}` is the same as `{send e1 mult e2}`, and `{select e1 e2}` is the same as `{send e1 select e2}`.

Conceptually, numbers such as `1` and `-1` are instances of a hidden built-in class. The `plus` and `mult` methods of the class recognize arguments that are instances of the same class and, in that case, they perform addition and muliplication in the usual way. When `plus` or `mult` recieves a value that is not an instance of the built-in number class, then it effectively swaps the argument order: it calls the `plus` or `mult` method of the given argument, passing along `this` as the new argument. (Of course, if two different classes use this strategy, then trying to add or multiply an instance of one class with an instance of the other will end up in an infinite loop.)

```
(test (interp-prog (list
                     '{class zero extends object
                             {}
                             {plus arg}
                             {mult this}
                             {select {send arg zero 0}}})
                    '{+ 7 {* 8 {new zero}}})
      '7)
(test (interp-prog (list
                     '{class infinity extends object
                             {}
                             {plus this}
                             {mult this}
                             {select {send arg nonzero 0}}})
                    '{+ 7 {new infinity}})
      `object)
(test (interp-prog (list
                     '{class infinity extends object
                             {}
                             {plus this}
                             {mult this}
                             {select {send arg nonzero 0}}}
                     '{class snowball extends object
                                  {size}
                                  {zero this}
                                  {nonzero {new snowball {+ 1 {get this size}}}}})
                    '{get {select {new infinity} {new snowball 3}} size})
      '4)
```

Last update: Tuesday, October 20th, 2015
mflatt@cs.utah.edu