Enhanced S3 Programming (Rough Draft)

Charlotte Maia

February 6, 2010

Abstract

This vignette introduces the oosp package, with a special emphasis on enhanced S3 programming.

Introduction

There are a number of class and object systems available for R, however the one that is by far the most prevalent is S3. Maybe because it's been around for a while, or maybe because it's simple... However, S3 (at least the kind of S3 we usually see) does impose a number of restrictions on R programmers. One of the key goals of the oosp package, is to support an enhanced kind of S3 programming and give R programmers all the flexibility that they never needed, and more...

Perhaps the biggest problem with S3, is for R package developers, who (if they don't what they're doing, and I've been hit by this too...) is R Check complaining about S3 methods that haven't been implemented properly, forcing one (at the R Check stage) to re-write functions and documentation. Let's say we have an object called point, with attributes x and y. Then let's say we wish to create a print method print.point (p), then the problem hits us...

This vignette looks at an alternative solution to the problem above, as well as considering some issues related to inheritance in S3. Other vignettes in this package consider other issues related to object oriented programming with S3.

Mask Functions

Let's consider the example above in more detail. We can create the class very easily, and at face value we can also create a print method very easily. Then creating and printing the object is trivial...

```
> #a possible class definition
> point = function (x, y) structure (list (x=x, y=y), class="point")


> #a possible print method
> print.point = function (p) cat ("x:", p$x, "\ny:", p$y, "\n")


> p = point (0, 0)
> p
x: 0
y: 0
```

At face value, if works fine. However, let's try and make a package...

```
> R CMD check My1stRPackage
```

```
* checking S3 generic/method consistency ... WARNING
print:
  function(x, ...)
print.point:
  function(p)
```

After a few changes...

```
> #another possible print method
> print.point = function (x, ...) cat ("x:", x$x, "\ny:", x$y, "\n")
```

Now, R Check is content, however I don't want to call my object x, I want to call p. So the oosp package implements mask functions, that "mask" a subset of the standard generics. In principle, we should still include the dots argument, however otherwise we can use what ever arguments we want.

Currently (these may change) the oosp package masks print, format, summary, plot, lines, predict, fitted, residuals, weights, as.list, as.data.frame and mean.

Now, if we load oosp, we can use p instead of x, and R Check is still content.

```
> #another possible print method
> print.point = function (p, ...) cat ("x:", p$x, "\ny:", p$y, "\n")
```

Extending Objects

A major concept in object oriented programming is inheritance. In S3, we don't really extend classes, we extend objects, however this really just an implementation issue (so we are going to pretend that we do extend classes).

The standard way of defining a class, is to set the class attribute, which we did earlier for our point class. For the case of inheritance, our class attribute is a character vector. So lets say we had a constructor for a circle class, that extended point, we might include a line such as:

```
> class (obj) = c ("circle", "point")
```

That's OK so far. However, lets say we defined another class, called discretecircle, that extends circle. We could include the following line in our constructor:

```
> class (obj) = c ("discretecircle", "circle", "point")
```

However, we have two problems (not big problems, however still problems). Firstly, this is verbose. For more classes, becomes more verbose. Secondly, if we decide to rename point as pos, then not only do we have to change the point and circle, constructors, we also have to change the discretecircle constructor as well, and again for more class, becomes troublesome.

If we assume that the superclass constructor is called before modifying the class attribute, then we can kind of solve these problems with something like:

```
> class (obj) = c ("discretecircle", class (obj) )
```

This is OK, however to make our life easier, the oosp package implements the function extend (which is nothing fancy). It takes an object, extends the class attribute, and returns the object). So the above line could be written as:

```
> obj = extend (obj, "discretecircle")
```

Which (in my opinion), is much nicer. Note that if this is the last line of the constructor, the assignment can be omitted (likewise, the other examples can be written using the structure function, as in the point example).