Use of and Using R as an Object Oriented Language

John James

john@mango-solutions.com

Why?

- Encapsulation
  - It is good to think in nouns and not verbs. Thinking in nouns forces a definition of what are the concepts of the problem. Verbs are about solutions. Run from these.
- Abstraction
  - The more generally we think about a problem the more we see its actual structure. The initial view we have of a problem is generally wrong and often muddled. Much problem solving is about determining the correct view to solve it.

Object Orientation

- Object-oriented programming is a style of programming that has become popular in recent years. Much of the popularity comes from the fact that it makes it easier to write and maintain complicated systems. It does this through several different mechanisms.

- Central to any object-oriented language are the concepts of class and of methods. A class is a definition of an object. Typically a class contains several slots that are used to hold class-specific information. An object in the language must be an instance of some class. Programming is based on objects or instances of classes.

- Computations are carried out via methods. Methods are basically functions that are specialized to carry out specific calculations on objects, usually of a specific class. This is what makes the language object oriented. In R, *generic functions are used to determine the appropriate method*. The generic function is responsible for determining the class of its argument(s) and uses that information to select the appropriate method.

http://cran.r-project.org/doc/manuals/R-lang.html

S3 Generic Methods

- Users can easily write their own methods and generic functions. A generic function is simply a function with a call to UseMethod. A method is simply a function that has been invoked via method dispatch. This can be as a result of a call to either UseMethod or NextMethod.
- It is worth remembering that methods can be called directly. That means that they can be entered without a call to UseMethod having been made and hence the special variables .Generic, .Class and .Method will not have been instantiated. In that case the default rules detailed above will be used to determine these.
- The most common use of generic functions is to provide print and summary methods for statistical objects, generally the output of some model fitting process. To do this, each model attaches a class attribute to its output and then provides a special method that takes that output and provides a nice readable version of it. The user then needs only remember that print or summary will provide nice output for the results of any analysis

S4 Generic Methods

- `setGeneric(name, def= , group=list(), valueClass=character(), where= , package= , signature= , useAsDefault= , genericFunction= , simpleInheritanceOnly = )`
- When a call to a generic function is evaluated, a method is selected corresponding to the classes of the actual arguments in the signature. s information about the actual classes.
- `setMethod(f, signature=character(), definition, where = topenv(parent.frame()), valueClass = NULL, sealed = FALSE)`

C++ Templates

- Function templates are special functions that can operate with *generic types*. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

```
template <class myType>
myType GetMax (myType a, myType b) { return
  (a>b?a:b);
 }
```

C++ Templates

- Class Templates are used where we have multiple copies of code for different data types with the same logic..

```
template  <typename T>
class MyQueue
{
        std::vector<Tdata;
    public:
        void Add(T const &d);
        void Remove();
        void Print();
};
```

What's the problem?

- Easy to create a class in R and encapsulate methods, and data

```
setClass("S1", representation(X ="numeric",
  I="integer"))
[1] "S1"
inst1 <- new("S1", x=pi, i=as.integer(12))
inst2 <- new("S1", x=log(10),
  i=as.integer(4))
```

- Default methods and functions need to be separately defined

```
inst1[1]
Error in inst1[1] : object of type 'S4' is
  not subsettable
```

Code Generators

method.skeleton("show", "s1")

```
setMethod("show",
    signature(object = "s1"),
    function (object)
    {
        stop("Need a definition for
  the method here")
    }
)
```

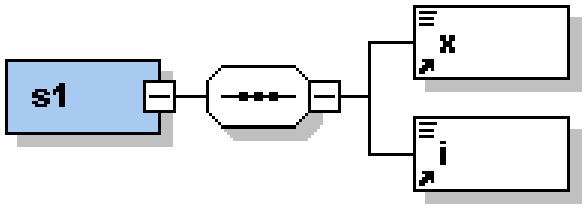
Example data

```
<?xml version="1.0" encoding="UTF-8"?>
<s1 xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance">
  <X>3.14159</X>
  <I>-11</I>
</S1>
```

XML Package

```
> s1 <- xmlRoot(xmlTreeParse('c:/My Eclipse/rplate/rplate/inst/xml/s1.xml'))
> xmlName(s1)
[1] "s1"
> xmlChildren(s1)[[1]]
<x>3.14159</x>
> makeClassTemplate(xmlChildren(s1)[[1]], types='numeric')
$name
[1] "x"

$slots
 text
"ANY"

$def
[1] "setClass('x',\n    representation('text' = 'ANY'))"

$coerce
[1] "setAs('XMLAbstractNode', 'x', function(from) xmlToS4(from))"

>
> new('x')
An object of class "x"
Slot "text":
NULL
```

Understand the data



rplate extension

- Work from the xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="X" type="xs:double"/>
 <xs:element name="I" type="xs:integer"/>
 <xs:element name="S1">
  <xs:complexType>
   <xs:sequence>
    <xs:element id="Real" ref="X"/>
    <xs:element id="Int" ref="I"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Using templates

- require(rplate)

- XSD <- xmlRoot(xmlTreeParse('c:/My Eclipse/rplate/rplate/inst/xml/S1.xsd'))

- xsdTemplate(node=xmlChildren(XSD)[[1]])
- xsdTemplate(node=xmlChildren(XSD)[[2]])
- xsdTemplate(node=xmlChildren(XSD)[[3]])

- s **<-** new('S1', Real=new('X', text=3.14))

- > s
- An object of class "S1"
- list()
- Slot "Real":
- An object of class "X"
- list()
- Slot "text":
- [1] 3.14
- Slot "Int":
- An object of class "I"
- list()

- > S4Toxml(s)
- <S1>
- <X>3.14</X>
- </S1>

Conclusions

- It is both useful and practical to generate R classes from a data definition
- Templates reduces the volume of R code that has the be defined.
- Checking of methods and programming is easier, as it produces readable code: the unreadable part is generated from data description developed within common tools
- Need: always use ‘setters’ (get, set) and (try to) hide ‘@’