S3 and S4 classes

Object Oriented Programming in R

Objects

It is an **abstraction**: Objects are something that have attributes (values) and behaviors (actions).

These are sometimes called states and methods. These are formally defined in the **object definition**.

We have two types of players: Monsters and Heroes.
For the hero character, we need to store the values of certain attributes (Table ??):

Heroes must also be able to certain behaviors (which we will call methods: These attributes and behaviors completely define the Hero.

Modules may be written that know how to interpret (interact with) heroes.

Table 9.1: Attributes of Hero characters.

Attribute	Value
Health	16
Strength	12
Agility	14
WeaponType	"mace"
ArmorType	"leather"

Table 9.2: Behaviors or methods of Hero characters.

Methods
move through the maze
attack monsters
pick up treasure

In R , What is an Objects ?

#R works on objects:

- Objects are ways of bundling parts of programs into small, manageable pieces.

- Objects are simply a definition for a type of data to be stored e.g., data vector, matrix, array, data frame, list, function

- An object is a component of a program that **knows how to perform certain actions and to interact with other pieces of the program**.

Cont..

- **Functions** can be described as ”black boxes” that take an input and spit out an output. Objects can be thought of as ”smart” black boxes.

That is, **objects can know how to do more than one specific task** (**method or behavior**), and they can store their own set of data.

Quick Summary

A class is a description of an things. A class can be defined using setClass( ) in the methods package.

An Object is an instance of a class. Objects can be created using new().

A method is a function that only operates on certain class of objects.

With Example

An object is a data structure having some attributes and methods which act on its attributes.

Class is a **blueprint for the object**. We can think of class like a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house.

House is the object. As, many houses can be made from a description, we can create many objects from a class.

An **object is also called an instance of a class** and the process of creating this object is called instantiation.

Generic Function

A generic function is an R function which dispatches methods. A generic function typically encapsulates a “generic” concept (e.g. plot, mean, predict, ...)

A method is the implementation of a generic function for an object of a particular class.

Two styles of Classes and Methods

S3 classes/Methods

Included with version-3 of the S Language.

Informal

Sometimes called old style classes/method

S4 classes and methods

More formal and rigorous

Included with S PLUS 6 and R 1.4.0 (introduced in DEC 2001)

Also called new style classes/Methods

S3 CLASS

S3 class is somewhat primitive in nature. It lacks a formal definition and object of this class can be created simply by adding a class attribute to it.

Example 1: S3 class

```
> # create a list with required components

> s <- list(name = "John", age = 21, GPA = 3.5)

> # name the class appropriately

> class(s) <- "student"
```

Above example creates a S3 class with the given list.

S4 CLASS

S4 class are an improvement over the S3 class. They have a formally defined structure which helps in making object of the same class look more or less similar.

Class components are properly defined using the `setClass()` function and objects are created using the `new()` function.

Example 2: S4 class

```
< setClass("student", slots=list(name="character", age="numeric", GPA="numeric"))
```

Reference Class - (not in syllabus)

It is more similar to the object oriented programming we are used to seeing in other major programming languages.

Reference classes are basically S4 classed with an environment added to it.

Example 3: Reference class

```
< setRefClass("student")
```

Demonstration of S3 class in R

S3 class

S3 class is the most popular and prevalent class in R programming language.

Most of the classes that come predefined in R are of this type. The fact that it is simple and easy to implement.

How to define S3 class and create S3 objects?

S3 class has no formal, predefined definition.

Basically, a list with its class attribute set to some class name, is an S3 object. The components of the list become the member variables of the object.

Following is a simple example of **how an S3 object of class student can be created**.

```
> # create a list with required components

> s <- list(name = "John", age = 21, GPA = 3.5)

> # name the class appropriately

> class(s) <- "student"

> # That's it! we now have an object of class "student"
```

Output

```
> s

$name

[1] "John"

$age

[1] 21

$GPA

[1] 3.5

attr(,"class")

[1] "student"
```

How to use constructors to create objects?

Tt is a good practice to use a function with the same name as class (not a necessity) to create objects.

This will bring some uniformity in the creation of objects and make them look similar. We can also add some integrity check on the member attributes this example we use the `attr()` **function to set the class attribute** of the object.

```
# a constructor function for the "student" class

student <- function(n,a,g) {

# we can add our own integrity checks

if(g>4 || g<0)  stop("GPA must be between 0 and 4")

value <- list(name = n, age = a, GPA = g)

# class can be set using class() or attr() function

attr(value, "class") <- "student"

value

  }
```

Here is a sample run where we create objects using this constructor.

```
> s <- student("Paul", 26, 3.7)

> s

$name

[1] "Paul"

$age

[1] 26

$GPA

[1] 3.7
```

```
attr(,"class")

[1] "student"

> class(s)

[1] "student"

> s <- student("Paul", 26, 5)

Error in student("Paul", 26, 5) : GPA must be between 0 and 4

> # these integrity check only work while creating the object using constructor

> s <- student("Paul", 26, 2.5)

> # it's up to us to maintain it or not

> s$GPA <- 5
```

Methods and Generic Functions

simply write the name of the object, its internals get printed.

In interactive mode, writing the name alone will print it using the `print()` function.

```
> s
$name
[1] "Paul"
$age
[1] 26
$GPA
[1] 3.7
attr(,"class")
[1] "student"
```

How to write your own method?

implement a method `print.student()` ourself.

```
print.student <- function(obj) {
cat(obj$name, "\n")
cat(obj$age, "years old\n")
cat("GPA:", obj$GPA, "\n")
  }
```

Now this method will be called whenever we `print()` an object of class "`student`".

In S3 system, methods do not belong to object or class, they belong to generic functions. This will work as long as the class of the object is set.

```
> # our above implemented method is called

> s

Paul

26 years old

GPA: 3.7

> # removing the class attribute will restore as previous

> unclass(s)

$name

[1] "Paul"

$age

[1] 26

$GPA

[1] 3.7
```

Demonstration of s4 classes

Overview

Unlike S3 classes and objects which lacks formal definition, we look at S4 class which is stricter in the sense that it has a formal definition and a uniform way to create objects.

How to define S4 Class?

S4 class is defined using the `setClass()` function.

In R terminology, member variables are called slots. While defining a class, we need to set the name and the slots (along with class of the slot) it is going to have.

Example 1: Definition of S4 class

```
setClass("student", slots=list(name="character", age="numeric", GPA="numeric"))
```

In the above example, we defined a new class called `student` along with three slots it's going to have `name`, `age` and `GPA`.

There are other optional arguments of `setClass()` which you can explore in the help section with `?setClass`.

How to create S4 objects?

S4 objects are created using the `new()` function.

Example 2: Creation of S4 object

```
> # create an object using new()
> # provide the class name and value for slots
> s <- new("student",name="John", age=21, GPA=3.5)
> s
An object of class "student"
Slot "name":
[1] "John"
Slot "age":
[1] 21
Slot "GPA":
   [1] 3.5
```

We can check if an object is an S4 object through the function `isS4()`.

```
> isS4(s)
```

```
  [1] TRUE
```

How to access and modify slot?

Just as components of a list are accessed using `$`, slot of an object are accessed using `@`.

Accessing slot

```
> s@name
```

```
[1] "John"
```

```
> s@GPA
```

```
[1] 3.5
```

```
> s@age
```

```
   [1] 21
```

Modifying slot directly

A slot can be modified through reassignment.

```
> # modify GPA

> s@GPA <- 3.7

> s

An object of class "student"

Slot "name":

[1] "John"

Slot "age":

[1] 21

Slot "GPA":
```

How to write your own method?

We can write our own method using `setMethod()` helper function.

For example, we can implement our class method for the `show()` generic as follows.

```
setMethod("show",
"student",
function(object) {
cat(object@name, "\n")
cat(object@age, "years old\n")
cat("GPA:", object@GPA, "\n")
}
  )
```

Now, if we write out the name of the object in interactive mode as before, the above code is executed.

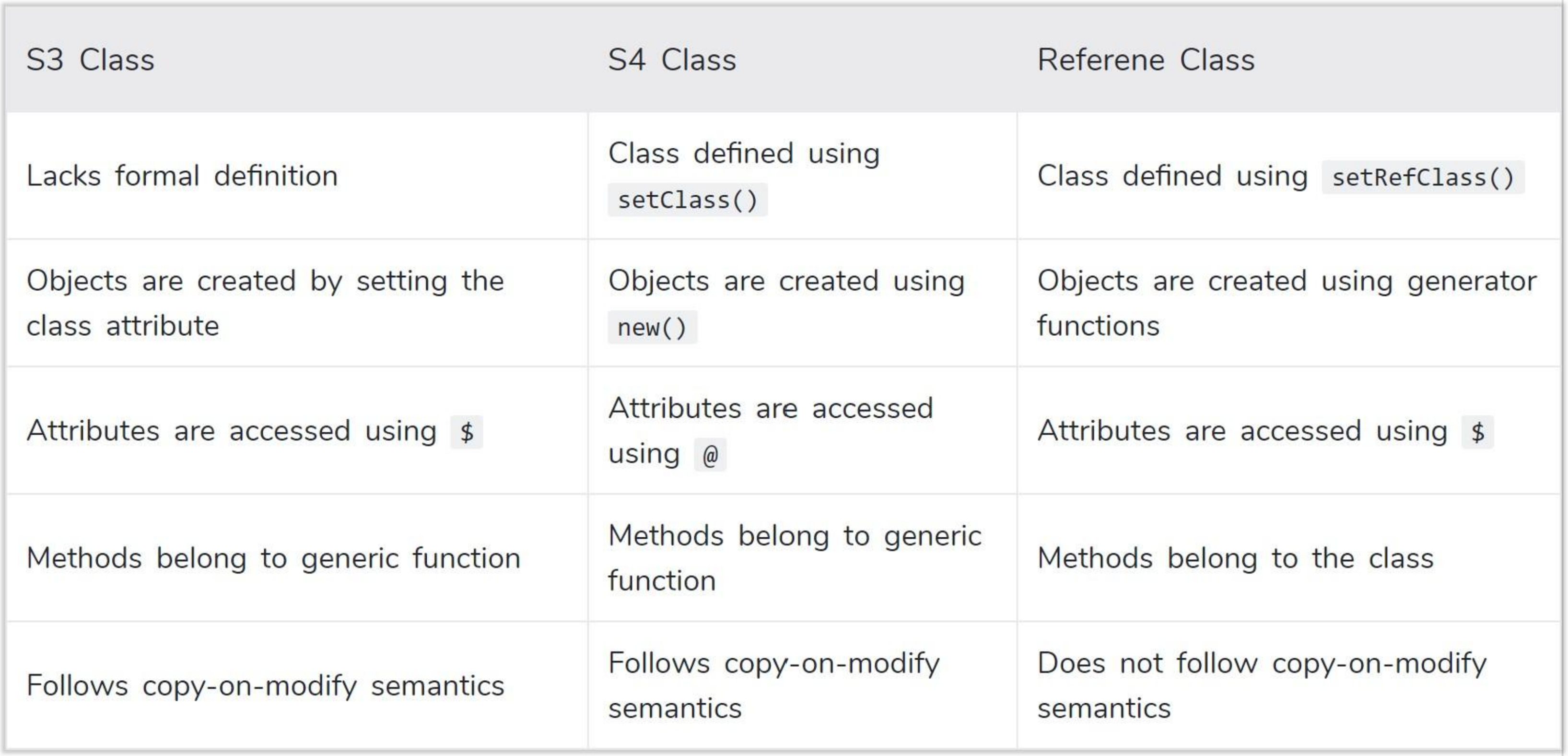
```
> s <- new("student",name="John", age=21, GPA=3.5)

> s    # this is same as show(s)

John

21 years old

GPA: 3.5
```



S3 Class	S4 Class	Referene Class
Lacks formal definition	Class defined using `setClass()`	Class defined using `setRefClass()`
Objects are created by setting the class attribute	Objects are created using `new()`	Objects are created using generator functions
Attributes are accessed using `$`	Attributes are accessed using `@`	Attributes are accessed using `$`
Methods belong to generic function	Methods belong to generic function	Methods belong to the class
Follows copy-on-modify semantics	Follows copy-on-modify semantics	Does not follow copy-on-modify semantics

Comparison between s4 class , s3 class and reference class.