CT5102: Programming for Data Analytics

Week 9:
Object Oriented Programming in R

https://github.com/JimDuggan/CT5102

Dr. Jim Duggan,
Information Technology,
School of Engineering & Informatics

Overview

- Central to any object-oriented system are the concepts of class and method
- A class defines the behaviour of objects by describing their attributes and their relationship to other classes
- R has three OO systems
  - S3
  - S4
  - Reference classes

Classes in R (lower case convention)

```
> v1<-1:10
> v1
 [1]  1  2  3  4  5  6  7  8  9 10
> class(v1)
[1] "integer"
> v2<-c(1.1,2.3,9.7)
> v2
[1] 1.1 2.3 9.7
> class(v2)
[1] "numeric"
> l<-list(v1,v2)
> str(l)
List of 2
 $ : int [1:10] 1 2 3 4 5 6 7 8 9 10
 $ : num [1:3] 1.1 2.3 9.7
> class(l)
[1] "list"
```

Creating an S3 object

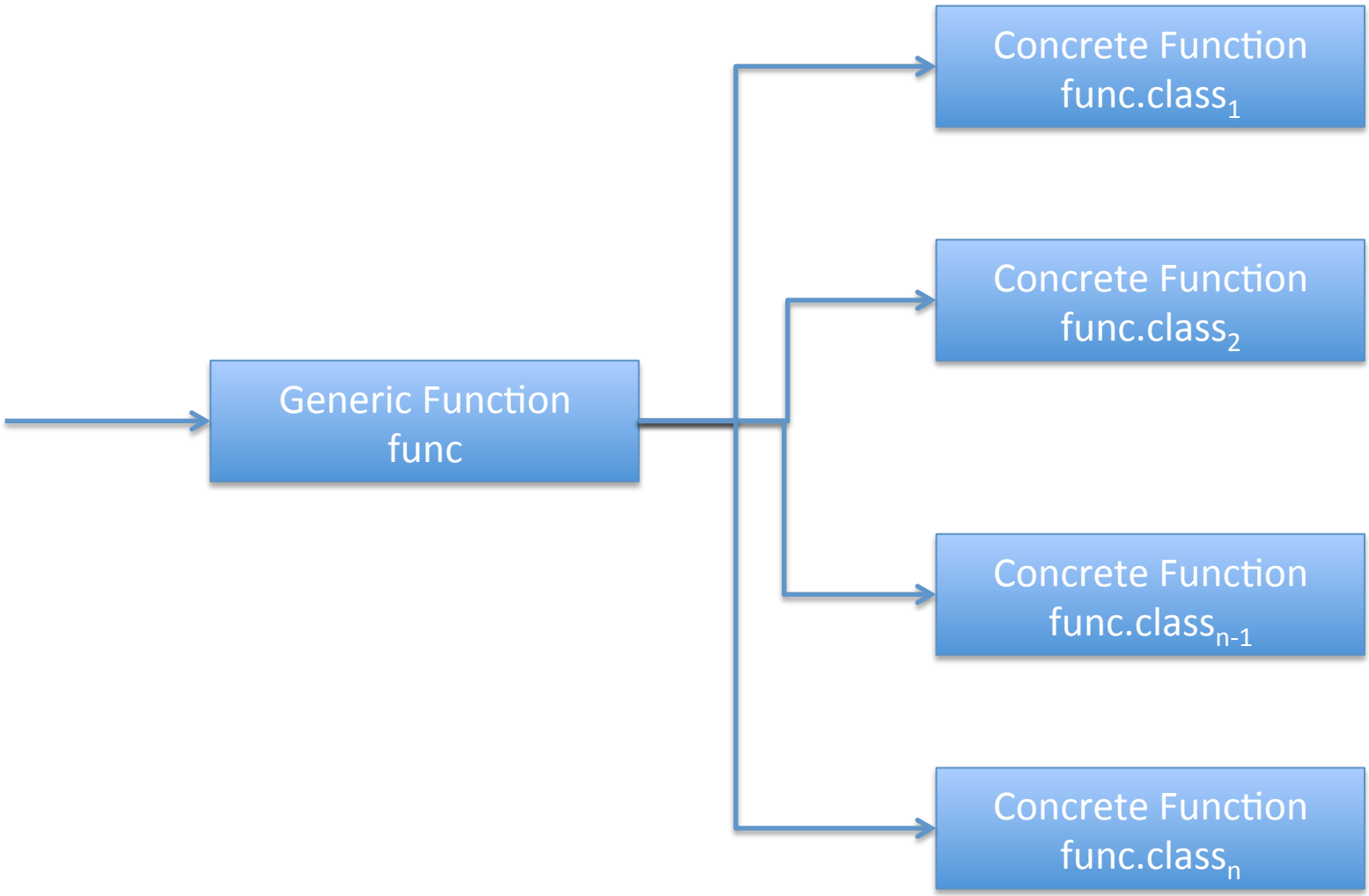
- Use the class() function
- Object as parameter
- Class type on RHS, as lowercase string
- Objects can belong to more than one class

```
> v1<-1:10
> v1
 [1]  1  2  3  4  5  6  7  8  9 10
> class(v1)<-"myclass"
> attributes(v1)
$class
[1] "myclass"

> v2<-1:10
> v2
 [1]  1  2  3  4  5  6  7  8  9 10
> class(v2)<-c(class(v2),"myclass")
> attributes(v2)
$class
[1] "integer" "myclass"
```

S3 Objects

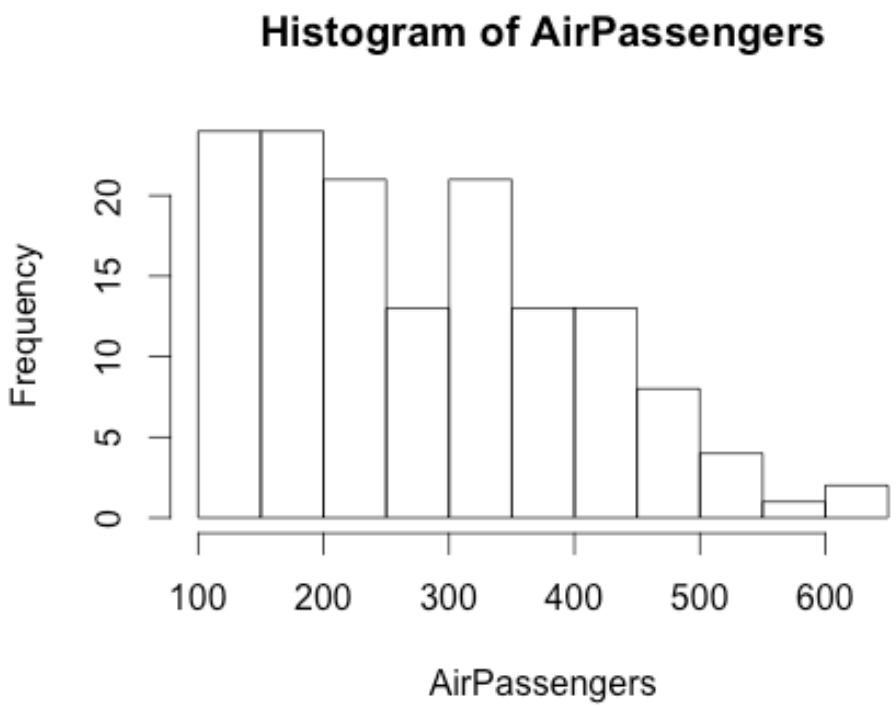
- Implements a style of programming known as *generic-function OO*
- Computations carried out by methods
- Contains a special kind of function – a *generic function* – that decides which function to call
- S3 – a very casual system, it has no formal definition of classes.

Overall Idea



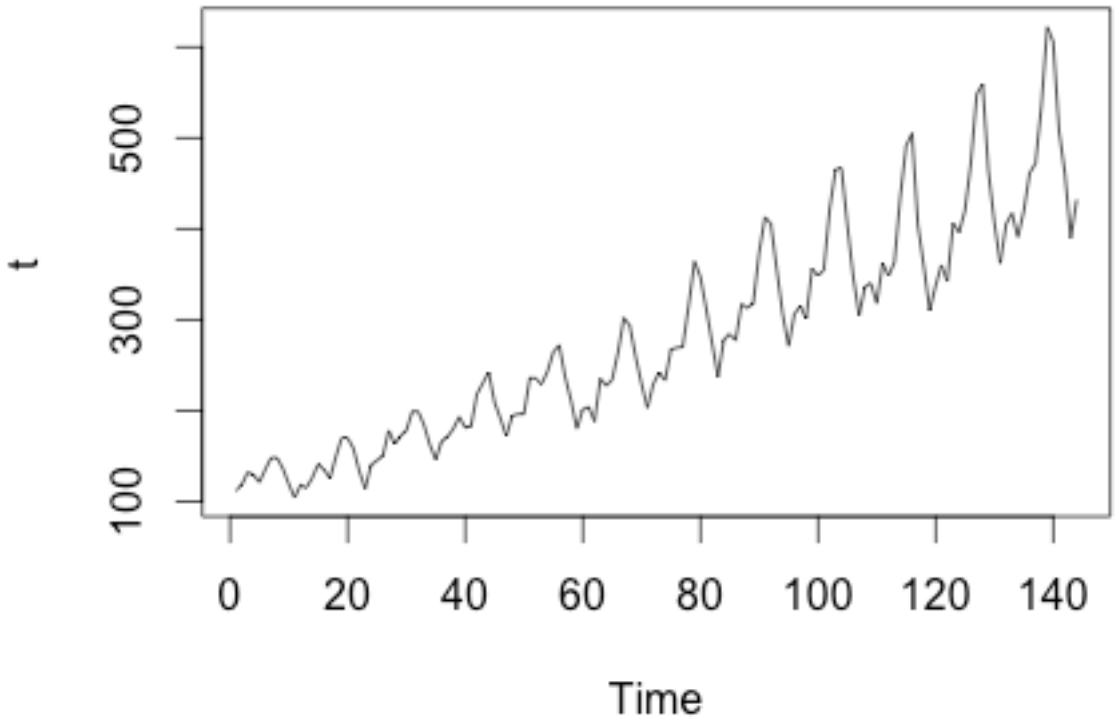
plot() produces histogram

```
> str(AirPassengers)
 Time-Series [1:144] from 1949 to 1961: 112 118 132 129 121 135 148 148
119 ...
> h<-hist(AirPassengers)
> attributes(h)
$names
[1] "breaks"   "counts"   "density"  "mids"     "xname"    "equidist"

$class
[1] "histogram"

> plot(h)
```



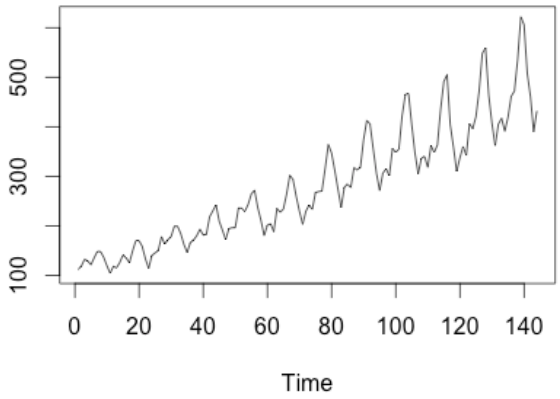
plot() produces time series

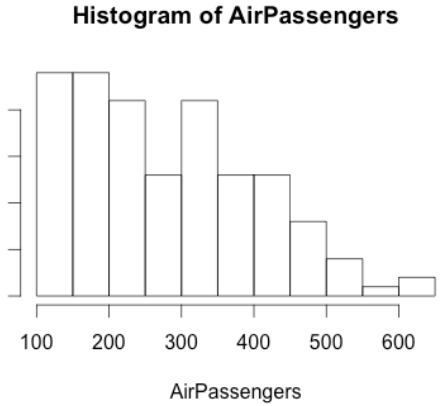
```
> t<-ts(AirPassengers)
> attributes(t)
$tsp
[1]   1 144   1

$class
[1] "ts"

> plot(t)
```



How..? Generic Functions.

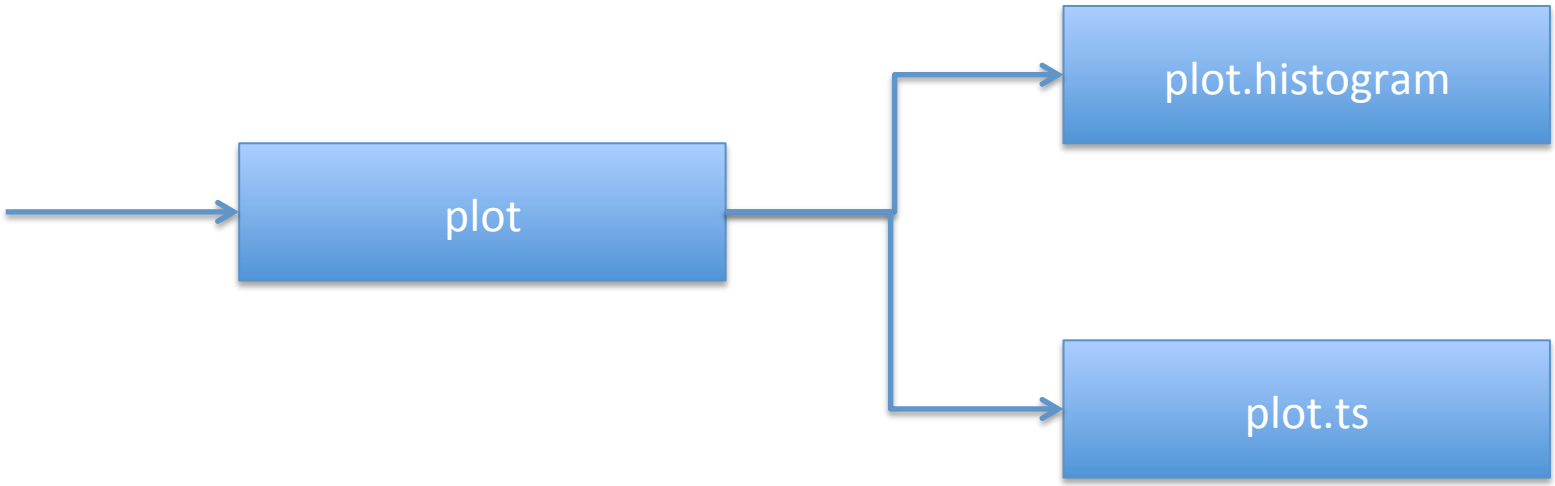
```
> plot
function (x, y, ...)
UseMethod("plot")
<bytecode: 0x10196e158>
<environment: namespace:graphics>
```






```
> methods("plot")
 [1] "plot.acf"             "plot.data.frame"     "plot.decomposed.ts"
 [4] "plot.default"         "plot.dendrogram"     "plot.density"
 [7] "plot.ecdf"            "plot.factor"         "plot.formula"
[10] "plot.function"        "plot.hclust"         "plot.histogram"
[13] "plot.HoltWinters"     "plot.isoreg"         "plot.lm"
[16] "plot.medpolish"       "plot.mlm"            "plot.ppr"
[19] "plot.prcomp"          "plot.princomp"       "plot.profile.nls"
[22] "plot.spec"            "plot.stepfun"        "plot.stl"
[25] "plot.table"           "plot.ts"             "plot.tskernel"
[28] "plot.TukeyHSD"
```

Overall idea…



Format for specific functions are *[generic function].[class name]*

Query methods for a class

```
> methods(class="ts")
 [1] "[.ts"              "[<-.ts"          "aggregate.ts"
 [4] "as.data.frame.ts" "cbind.ts"        "cycle.ts"
 [7] "diff.ts"           "diffinv.ts"      "kernapply.ts"
[10] "lines.ts"          "monthplot.ts"    "na.omit.ts"
[13] "Ops.ts"            "plot.ts"         "print.ts"
[16] "t.ts"              "time.ts"         "window.ts"
[19] "window<-.ts"
```

```
> methods(class="list")
[1] "all.equal.list"      "as.data.frame.list" "relist.list"
[4] "within.list"
```

Class Methods

Description

R possesses a simple generic function mechanism which can be used for an object-oriented style of programming. Method dispatch takes place based on the class(es) of the first argument to the generic function or of the object supplied as an argument to `UseMethod` or `NextMethod`.

Usage

```
UseMethod(generic, object)
```

Arguments

`generic`	a character string naming a function (and not a built-in operator). Required for `UseMethod`.
`object`	for `UseMethod`: an object whose class will determine the method to be dispatched. Defaults to the first argument of the enclosing function.
`...`	further arguments to be passed to the next method.

Challenge 9.1

```
> v1<-1:10
> v1
 [1]  1  2  3  4  5  6  7  8  9 10
> class(v1)<-"myclass"
> attributes(v1)
$class
[1] "myclass"
```

- Create a new print function for an object of type “myclass”
- This should print the value and also a message indicating that the function is being called
- How does the function get called?

Challenge 9.2

```
> v1<-1:10
> v1
 [1]  1  2  3  4  5  6  7  8  9 10
> class(v1)<-"myclass"
> attributes(v1)
$class
[1] "myclass"
```

- Is mean() a generic function?
- Implement a mean function for objects of “myclass”, with an appropriate information message
- What happens if you unclass v1 and call the mean?

Challenge 9.3

- Write a generic function called info.
- This should dispatch a call to an appropriate function that can deal with a list or a data frame
- What would happen if you wrote a function print.data.frame()?

```
> df<-data.frame(c=c(1,2,3))
> class(df)
[1] "data.frame"
> l<-list(c=c(2,3,4))
> class(l)
[1] "list"
```

Writing S3 Classes

- A class can be created by forming a list, where the components of the list are the member variables of the class

```
> a1<-list(number="1234",balance=200.98)
> class(a1)<-"account"
> a1
$number
[1] "1234"

$balance
[1] 200.98

attr(,"class")
[1] "account"
```

structure() function

```
> a2<-structure(list(number="5678", balance=100.23),
+                 class="account")
> a2
$number
[1] "5678"

$balance
[1] 100.23

attr(,"class")
[1] "account"
```

Creating a constructor function

```
accountFactory<-function(id, bal){
  structure(list(number=id,balance=bal),
            class="account")
}
```

```
> a1<-accountFactory("1111",200)
> str(a1)
List of 2
 $ number : chr "1111"
 $ balance: num 200
 - attr(*, "class")= chr "account"
```

```
> attributes(a1)
$names
[1] "number"  "balance"

$class
[1] "account"
```

Adding a print function

```
print.account<-function(x){
  cat("Printing account information...\n")
  cat("Customer Number = ",x$number,"\n")
  cat("Customer Balance = ",x$balance,"\n")
}
```

```
> a1<-accountFactory("12345678",200.21)
> print(a1)
Printing account information...
Customer Number =  12345678
Customer Balance =  200.21
```

Inheritance with S3 Classes

- The idea of inheritance is to form new classes as specialised version of old ones
- Our new class will inherit the methods of the old one

```
currentAccountFactory<-function(id, bal,int=0.0){
  structure(list(number=id,balance=bal,interest=int),
            class=c("currentaccount","account"))
}
```

R's function search mechanism

```
currentAccountFactory<-function(id, bal,int=0.0){
  structure(list(number=id,balance=bal,interest=int),
            class=c("currentaccount","account"))
}
```

UseMethod first searches "currentaccount" for a print method. When this fails, it tries "account"

```
> c1<-currentAccountFactory("6541111",100.65,2.4)
> print(c1)
Printing account information...
Customer Number =  6541111
Customer Balance =  100.65
```

Challenge 9.4

- Add debit and credit functions to the class
- These should also record each successive transaction value, and store these in a list, prefixed by either “DR” or “CR”
- Generic functions should be used for debit and credit
- The print function should be updated to print out the transactions