CT5102: Programming for Data Analytics

Week 10: Object Oriented Programming in R

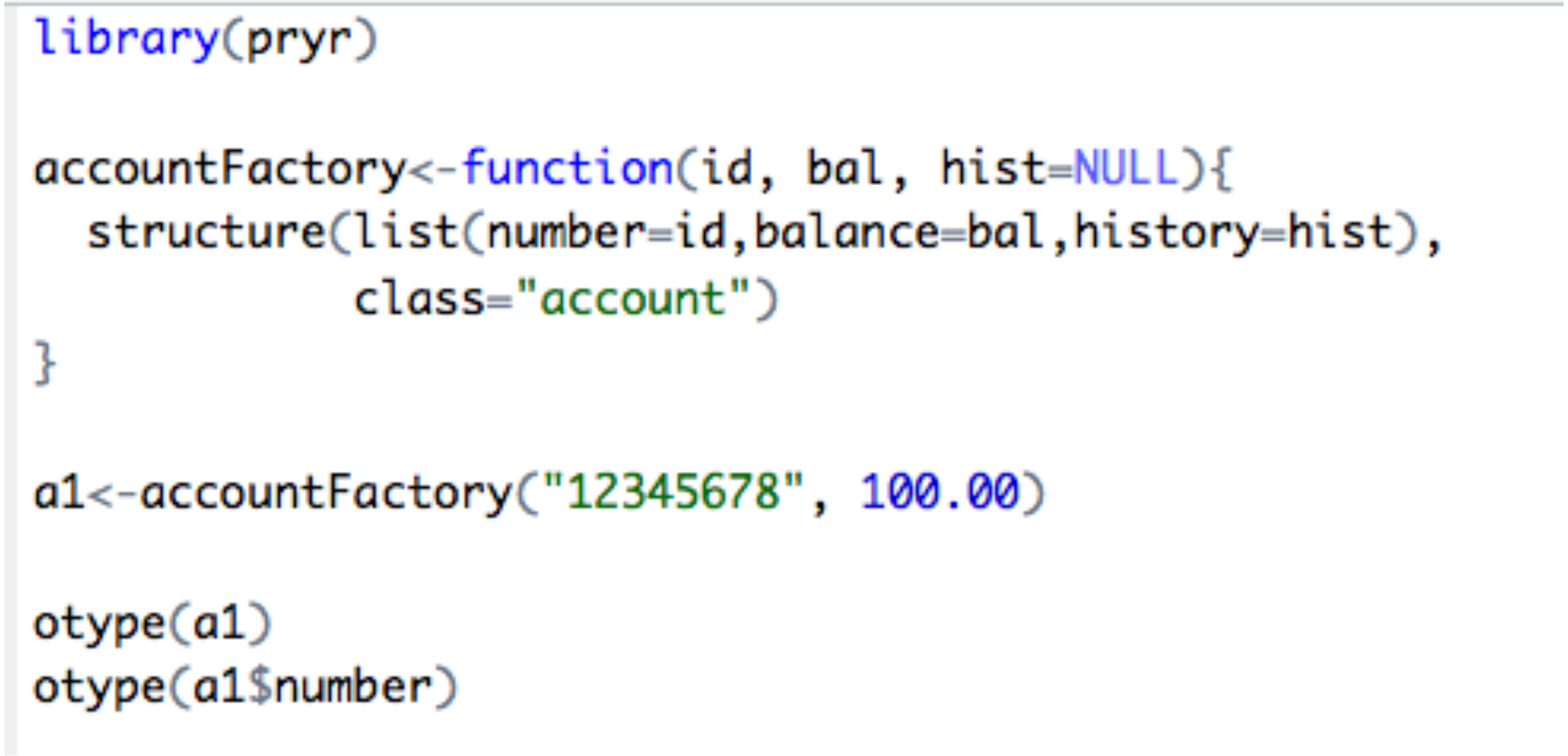
https://github.com/JimDuggan/CT5102

Dr. Jim Duggan,

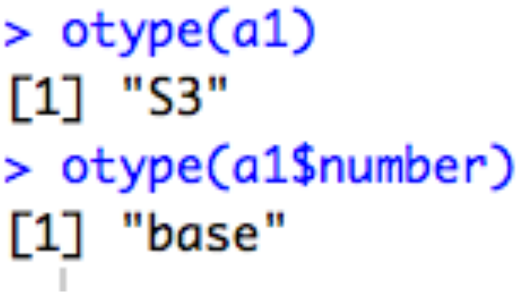
Information Technology,

School of Engineering & Informatics

Recognising objects



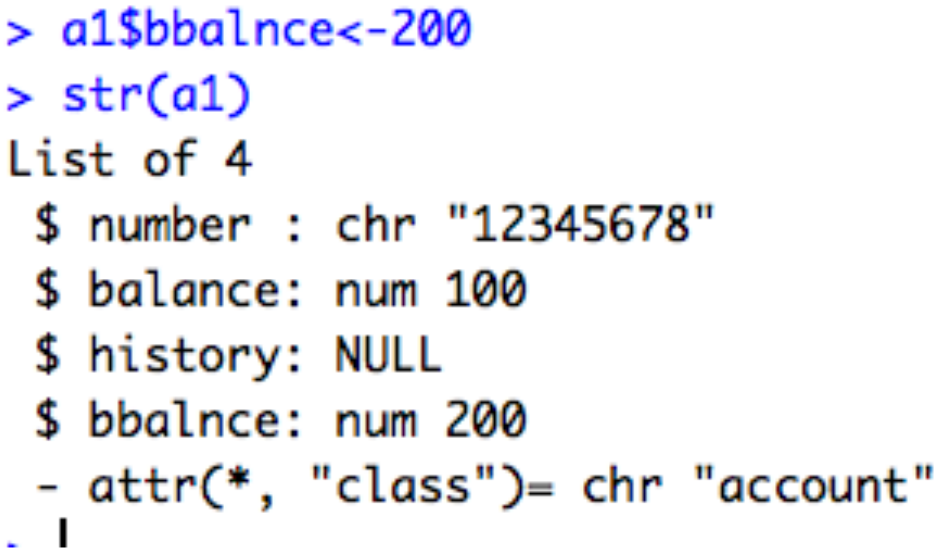
```
library(pryr)

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

a1<-accountFactory("12345678", 100.00)

otype(a1)
otype(a1$number)
```



```
> otype(a1)
[1] "S3"
> otype(a1$number)
[1] "base"
```

S3 Summary

- “Some programmers feel that S3 does not provide the safety associated with OOP.” (Matloff 2011)
- With S3, R will not complain with typos such as →
- The goal of S4 is to provide a more formal structure for objects



```
> a1$bbalnce<-200
> str(a1)
List of 4
 $ number : chr "12345678"
 $ balance: num 100
 $ history: NULL
 $ bbalnce: num 200
 - attr(*, "class")= chr "account"
```

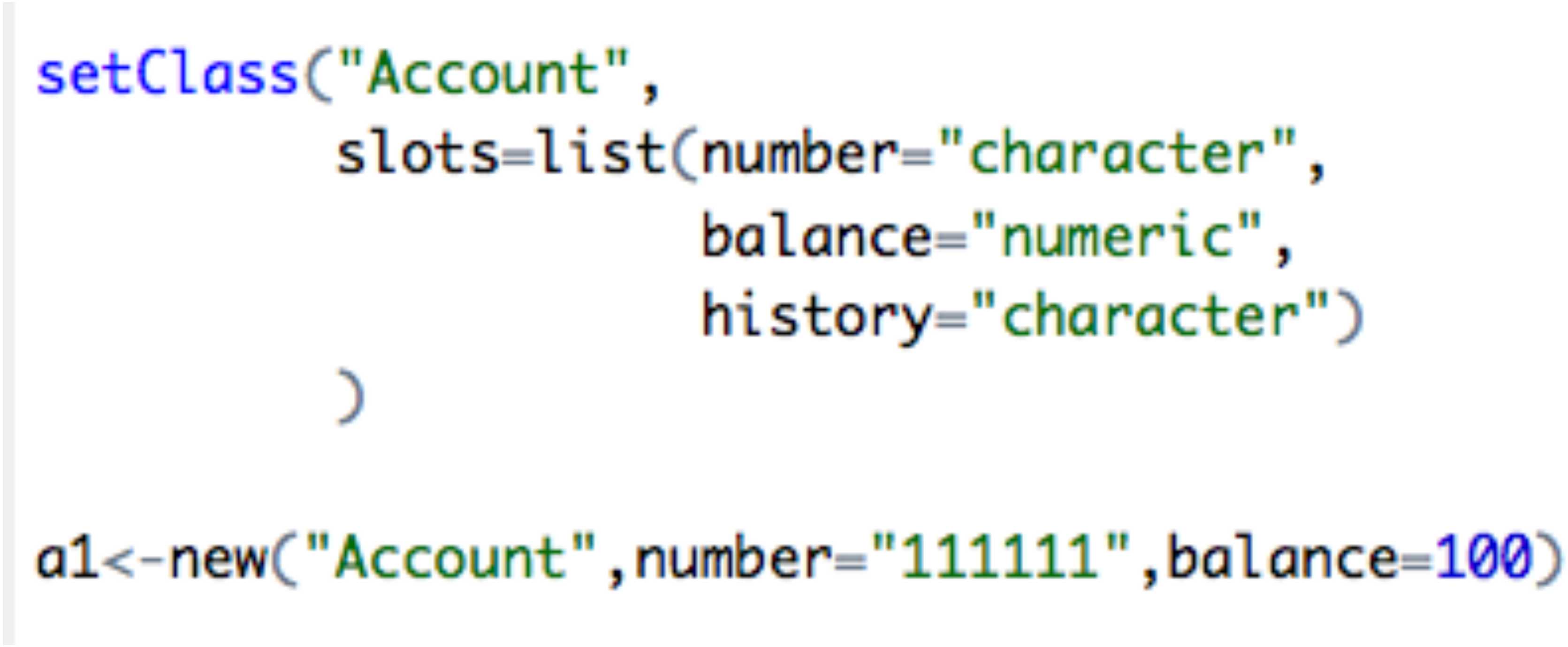
Comparison of S3 and S4

Operation	S3	S4
Define class	Implicit in constructor code	setClass()
Create object	Build list, set class attribute	new()
Reference member variable	$	@
Implement generic f	Define f.classname	setMethod
Declare generic	UseMethod()	setGeneric()

Defining S4 classes

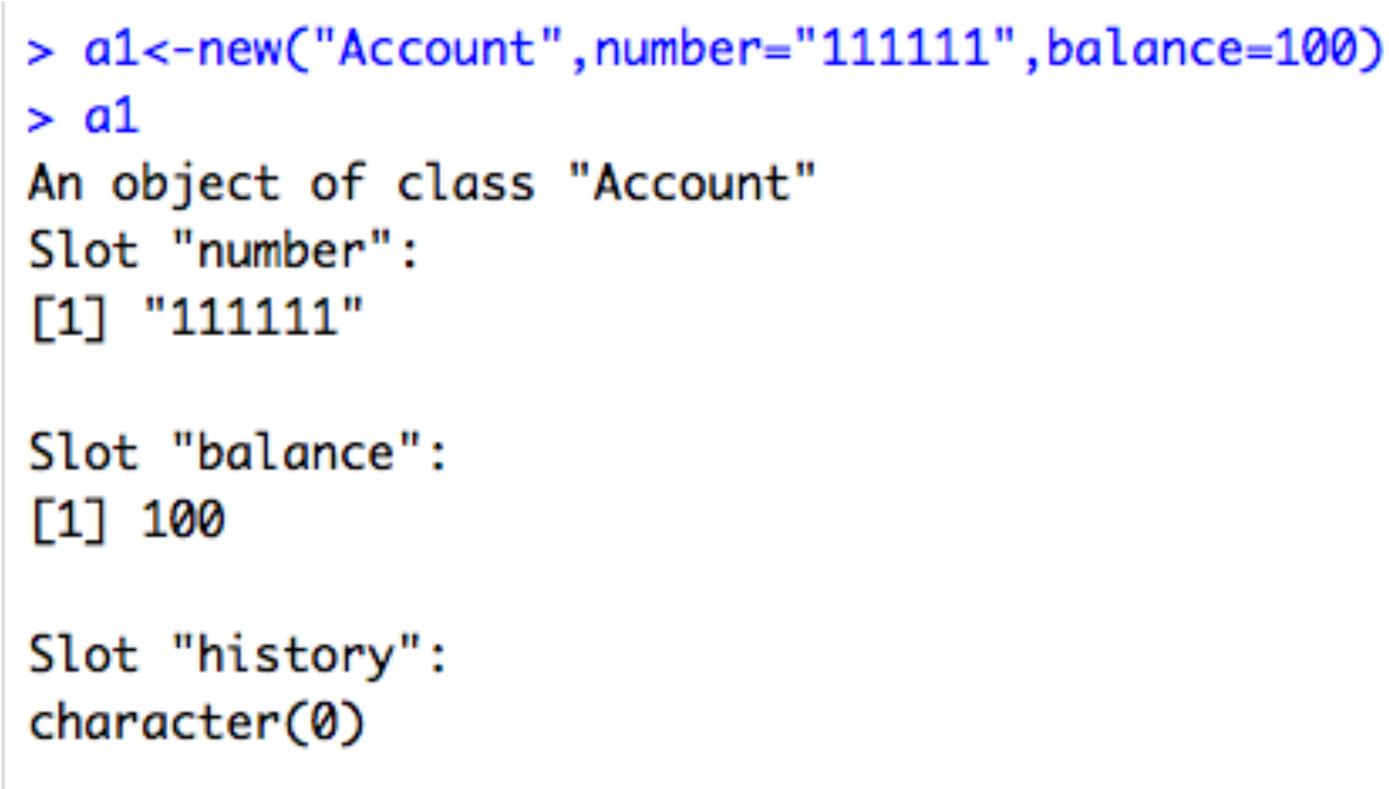
- An S4 class has three key properties (Wickham 2015)
  - A **name**: an alpha-numeric class identifier. By convention use *UpperCamelCase*
  - A named list of **slots** (fields), which defines slot names and permitted classes.
  - A string giving the class it inherits from, or in S4 terminology, that it **contains.**

Creating a class



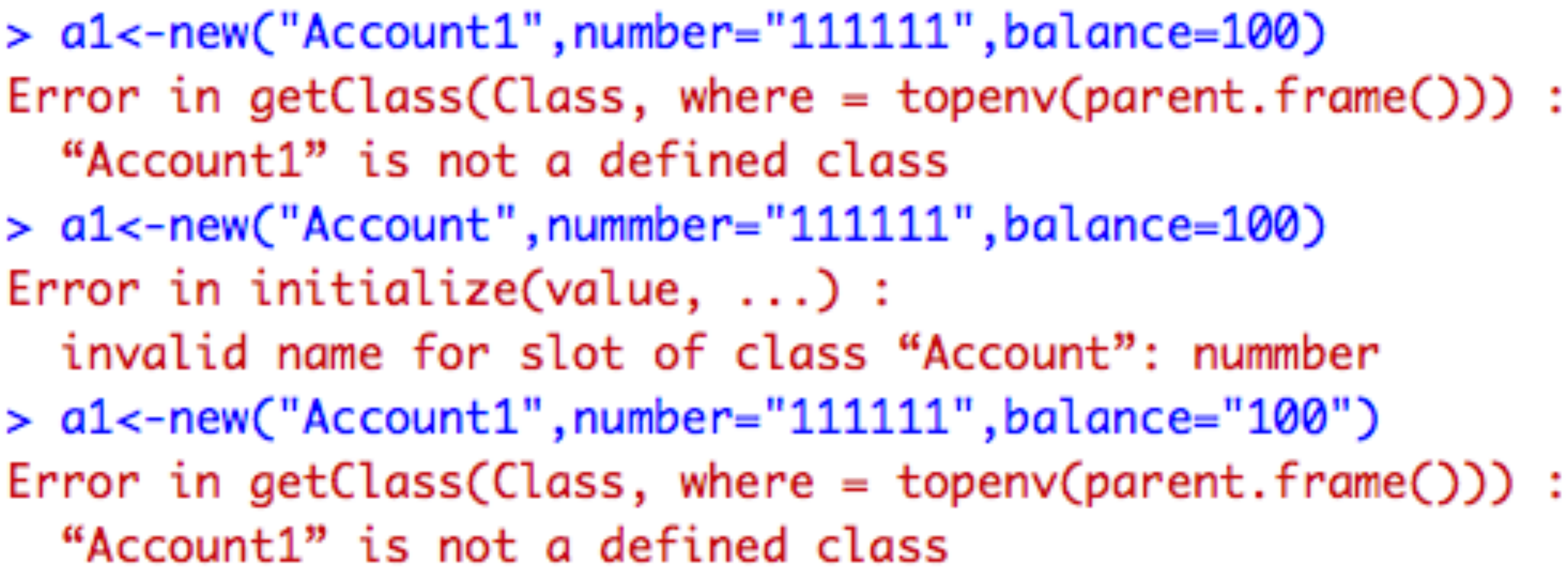
```
setClass("Account",
         slots=list(number="character",
                    balance="numeric",
                    history="character")
         )

a1<-new("Account",number="111111",balance=100)
```

Instantiating an object



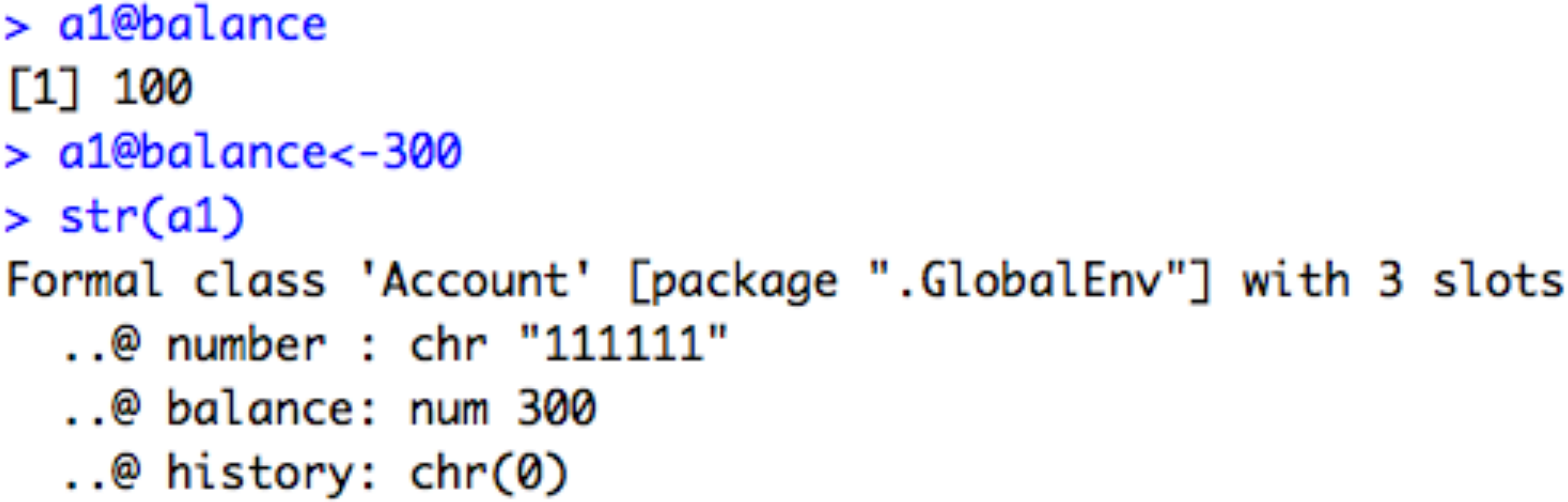
```
> a1<-new("Account",number="111111",balance=100)
> a1
An object of class "Account"
Slot "number":
[1] "111111"

Slot "balance":
[1] 100

Slot "history":
character(0)
```

Inbuilt checks in S4



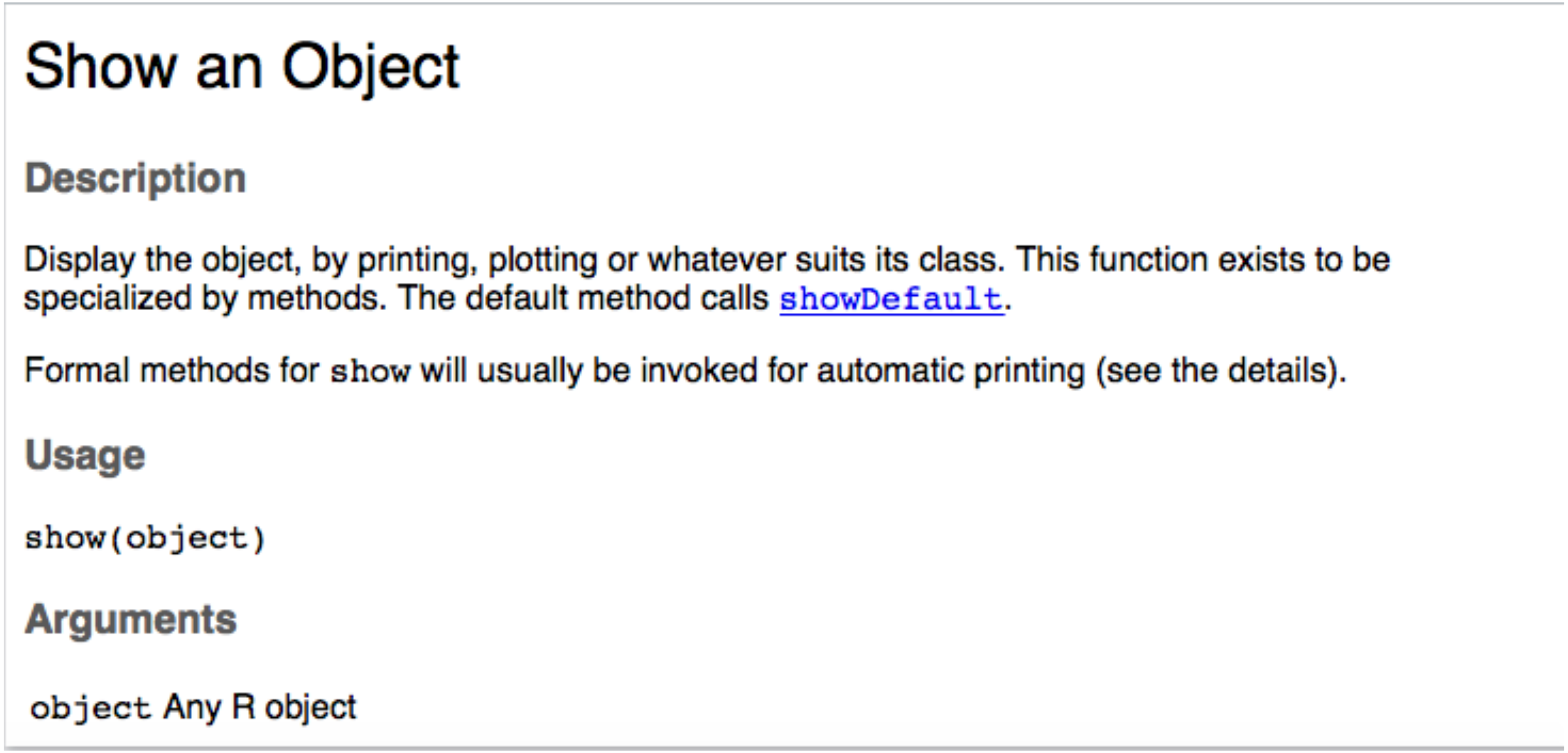
```
> a1<-new("Account1",number="111111",balance=100)
Error in getClass(Class, where = topenv(parent.frame())) :
  “Account1” is not a defined class
> a1<-new("Account",nummber="111111",balance=100)
Error in initialize(value, ...) :
  invalid name for slot of class “Account”: nummber
> a1<-new("Account1",number="111111",balance="100")
Error in getClass(Class, where = topenv(parent.frame())) :
  “Account1” is not a defined class
```

Accessing slots on an object



```
> a1@balance
[1] 100
> a1@balance<-300
> str(a1)
Formal class 'Account' [package ".GlobalEnv"] with 3 slots
  ..@ number : chr "111111"
  ..@ balance: num 300
  ..@ history: chr(0)
```

show() in R (for S4 classes)



Show an Object

Description

Display the object, by printing, plotting or whatever suits its class. This function exists to be specialized by methods. The default method calls `showDefault`.

Formal methods for `show` will usually be invoked for automatic printing (see the details).

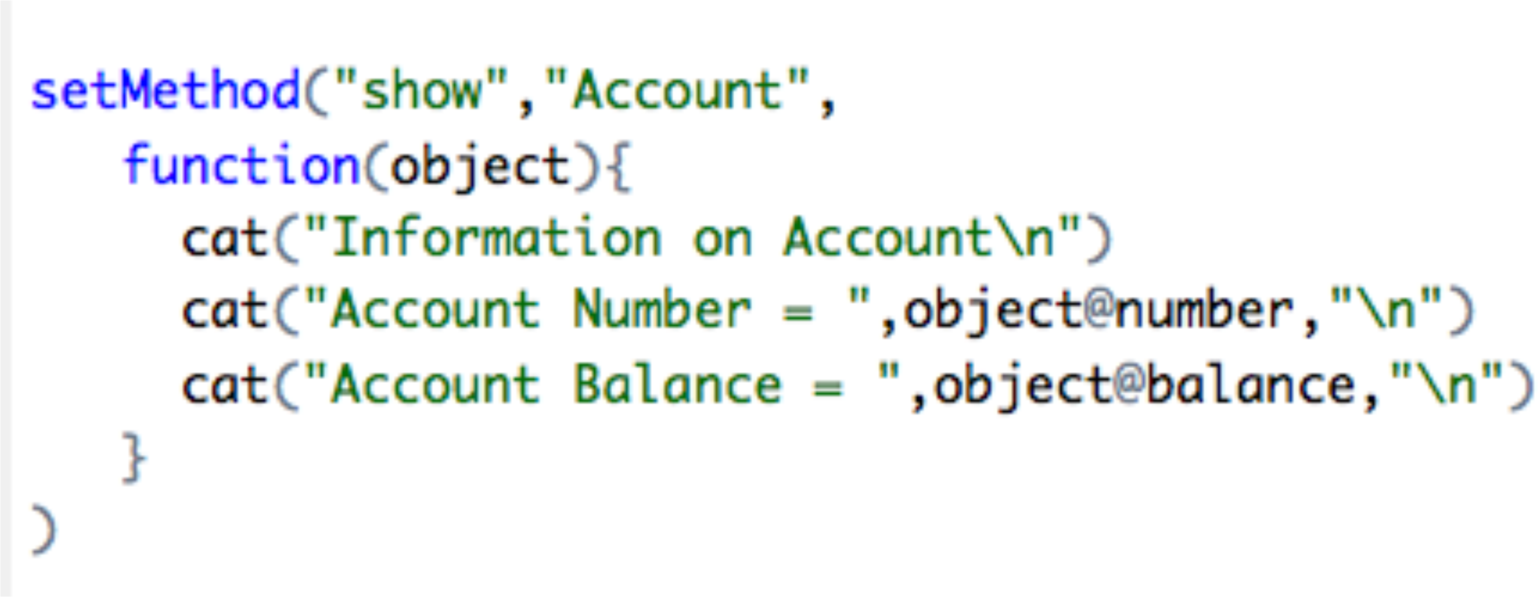
Usage

```
show(object)
```

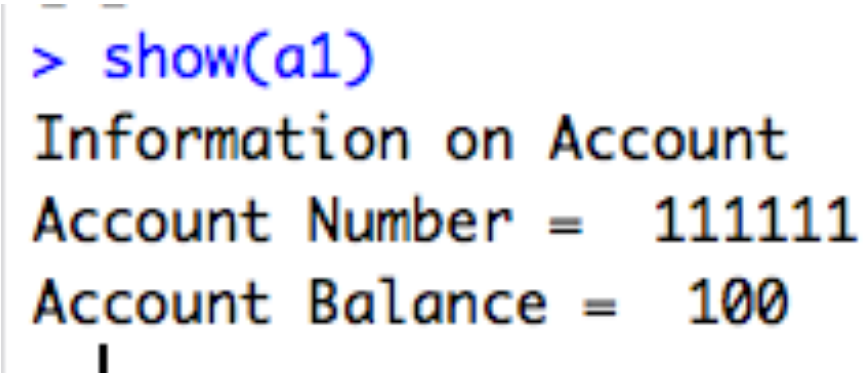
Arguments

`object` Any R object

Implementing a generic function in S4



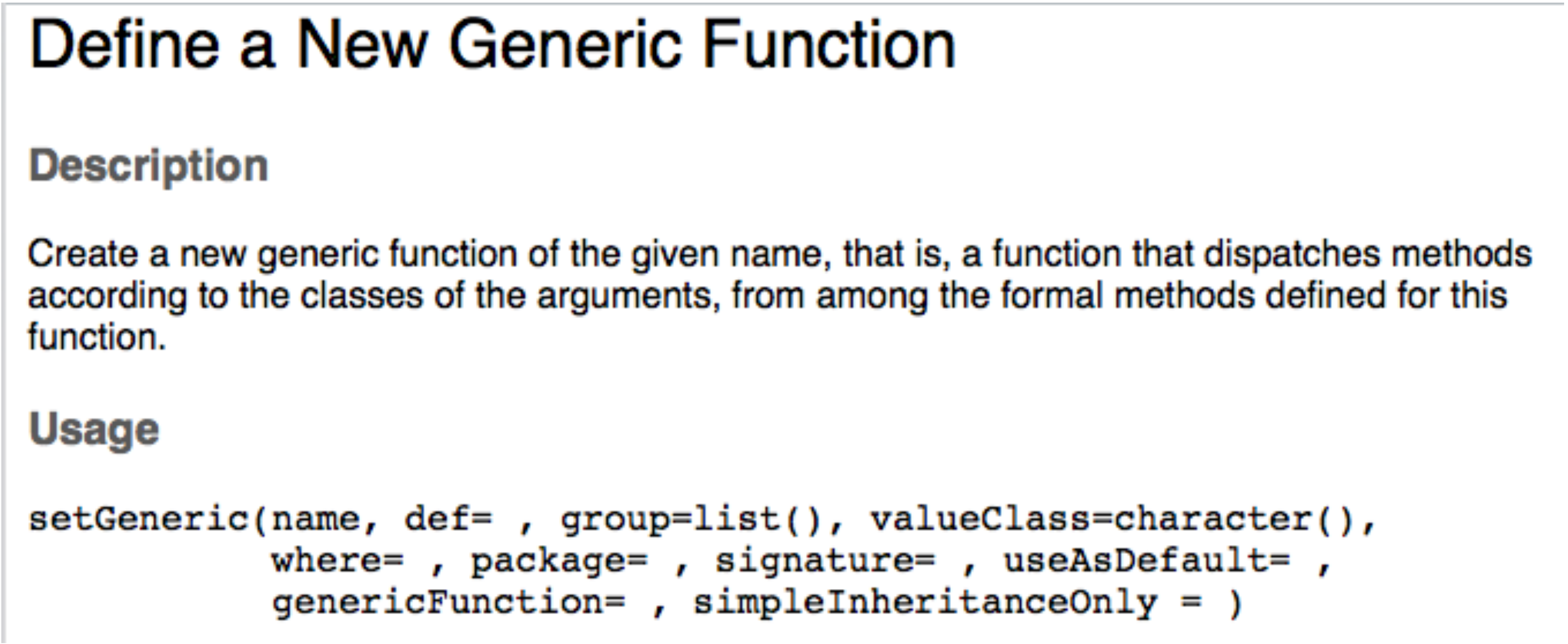
```
setMethod("show","Account",
   function(object){
     cat("Information on Account\n")
     cat("Account Number = ",object@number,"\n")
     cat("Account Balance = ",object@balance,"\n")
   }
)
```



```
> show(a1)
Information on Account
Account Number =  111111
Account Balance =  100
```

Writing a method for a class

Need to define a generic function first…



Define a New Generic Function

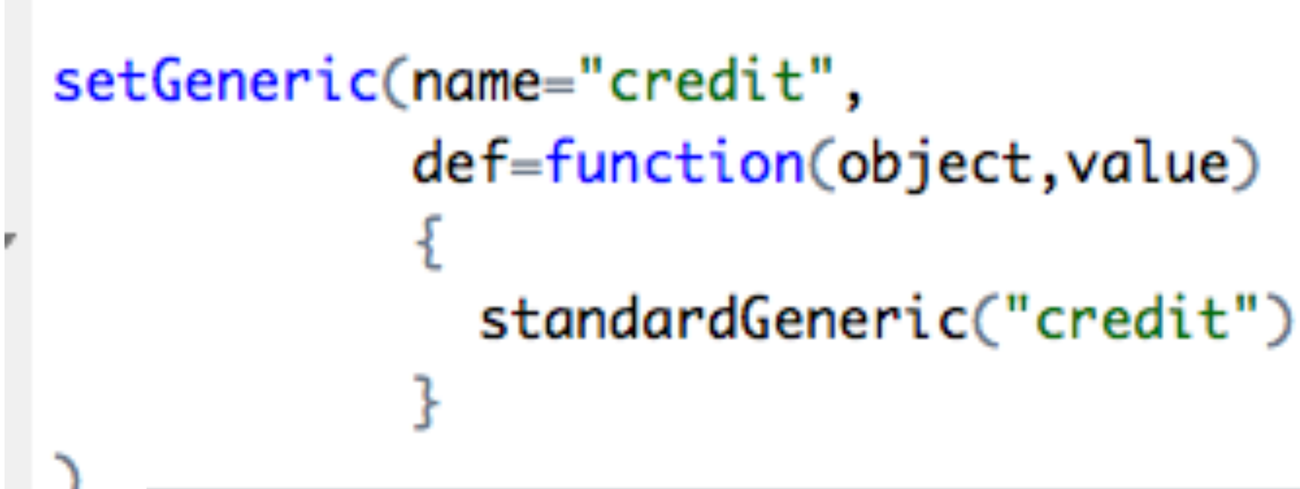
Description

Create a new generic function of the given name, that is, a function that dispatches methods according to the classes of the arguments, from among the formal methods defined for this function.

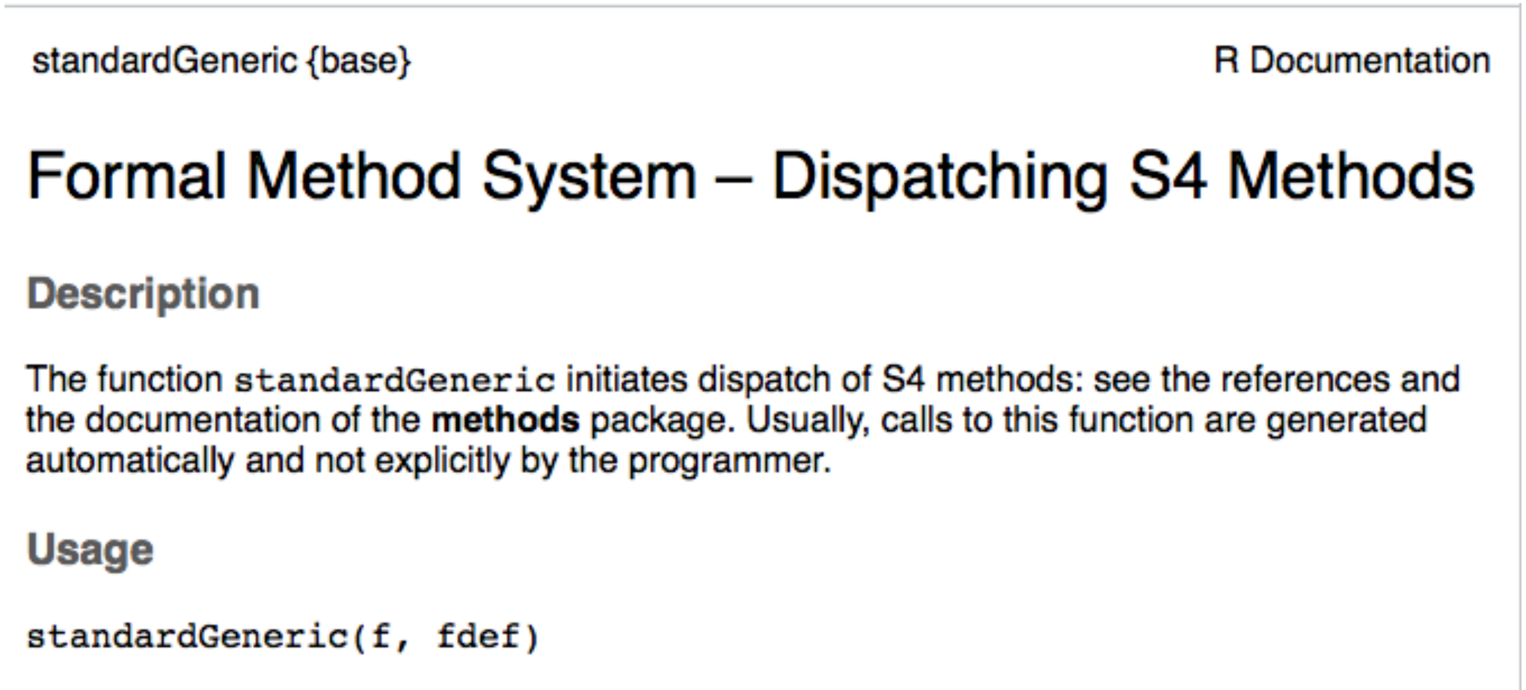
Usage

```
setGeneric(name, def= , group=list(), valueClass=character(),
           where= , package= , signature= , useAsDefault= ,
           genericFunction= , simpleInheritanceOnly = )
```

credit an account… generic function



```
setGeneric(name="credit",
           def=function(object,value)
           {
             standardGeneric("credit")
           }
)
```



standardGeneric {base}　　R Documentation

Formal Method System – Dispatching S4 Methods

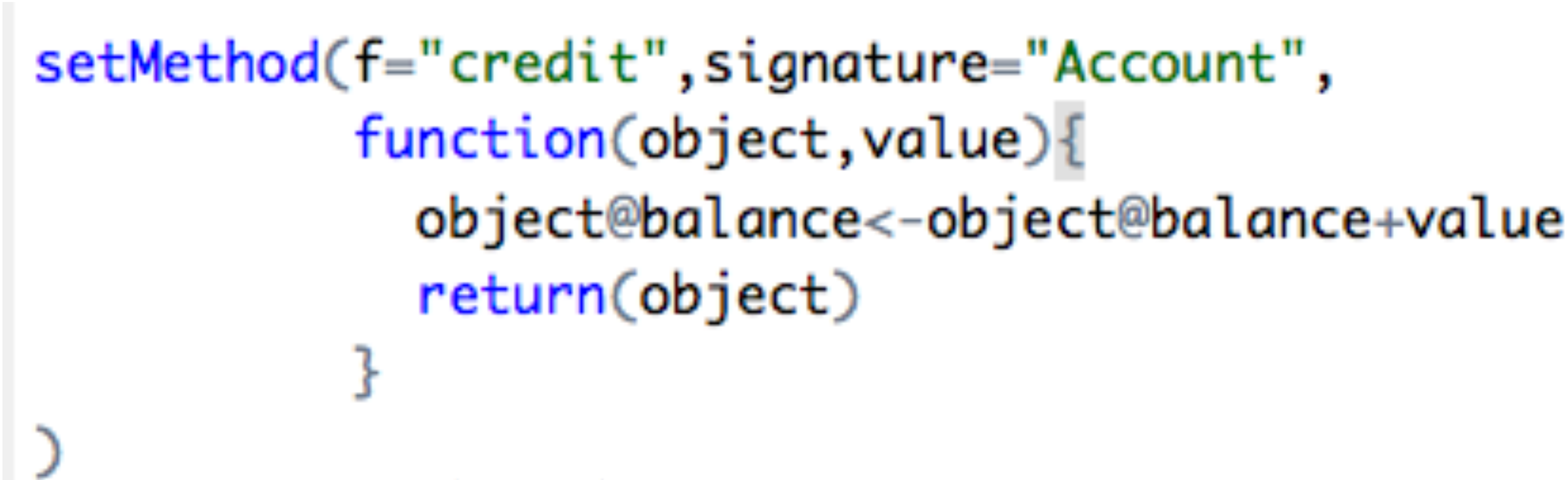
Description

The function `standardGeneric` initiates dispatch of S4 methods: see the references and the documentation of the **methods** package. Usually, calls to this function are generated automatically and not explicitly by the programmer.

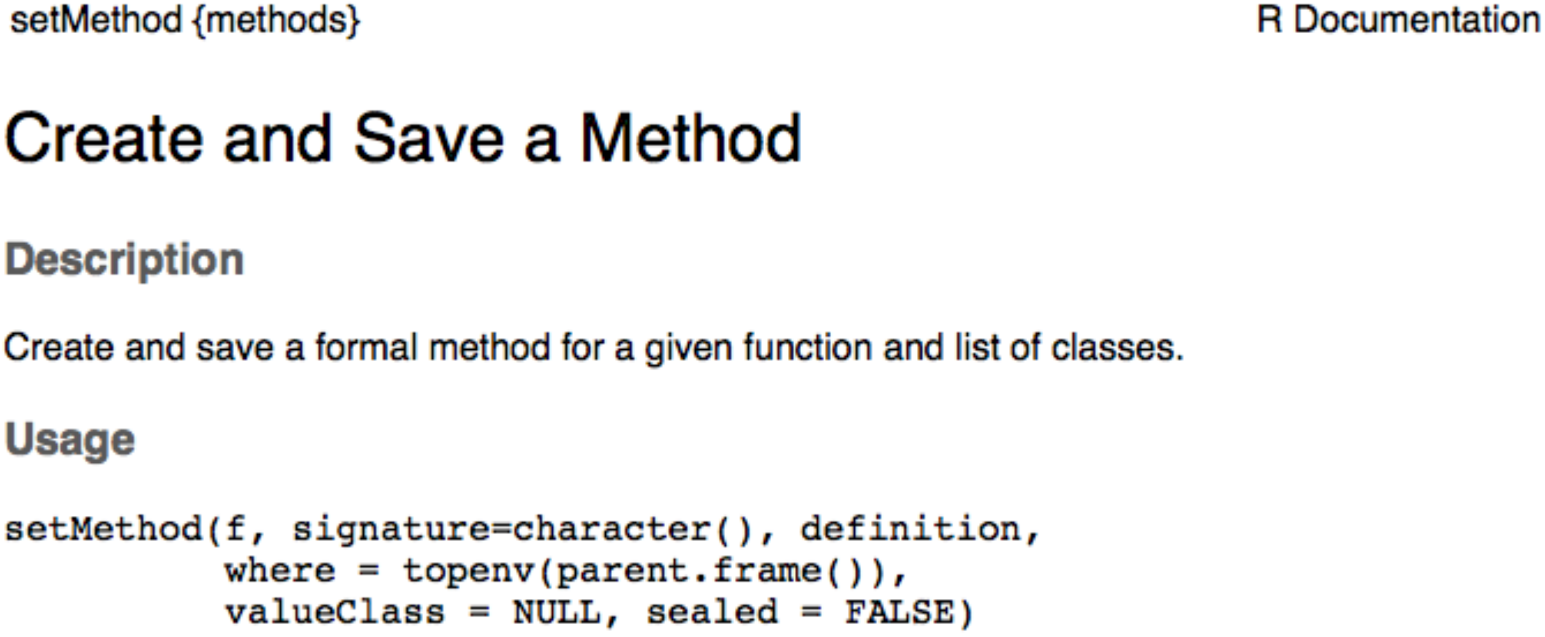
Usage

```
standardGeneric(f, fdef)
```

Writing the “concrete” method



```
setMethod(f="credit",signature="Account",
          function(object,value){
            object@balance<-object@balance+value
            return(object)
          }
)
```



setMethod {methods} R Documentation

Create and Save a Method

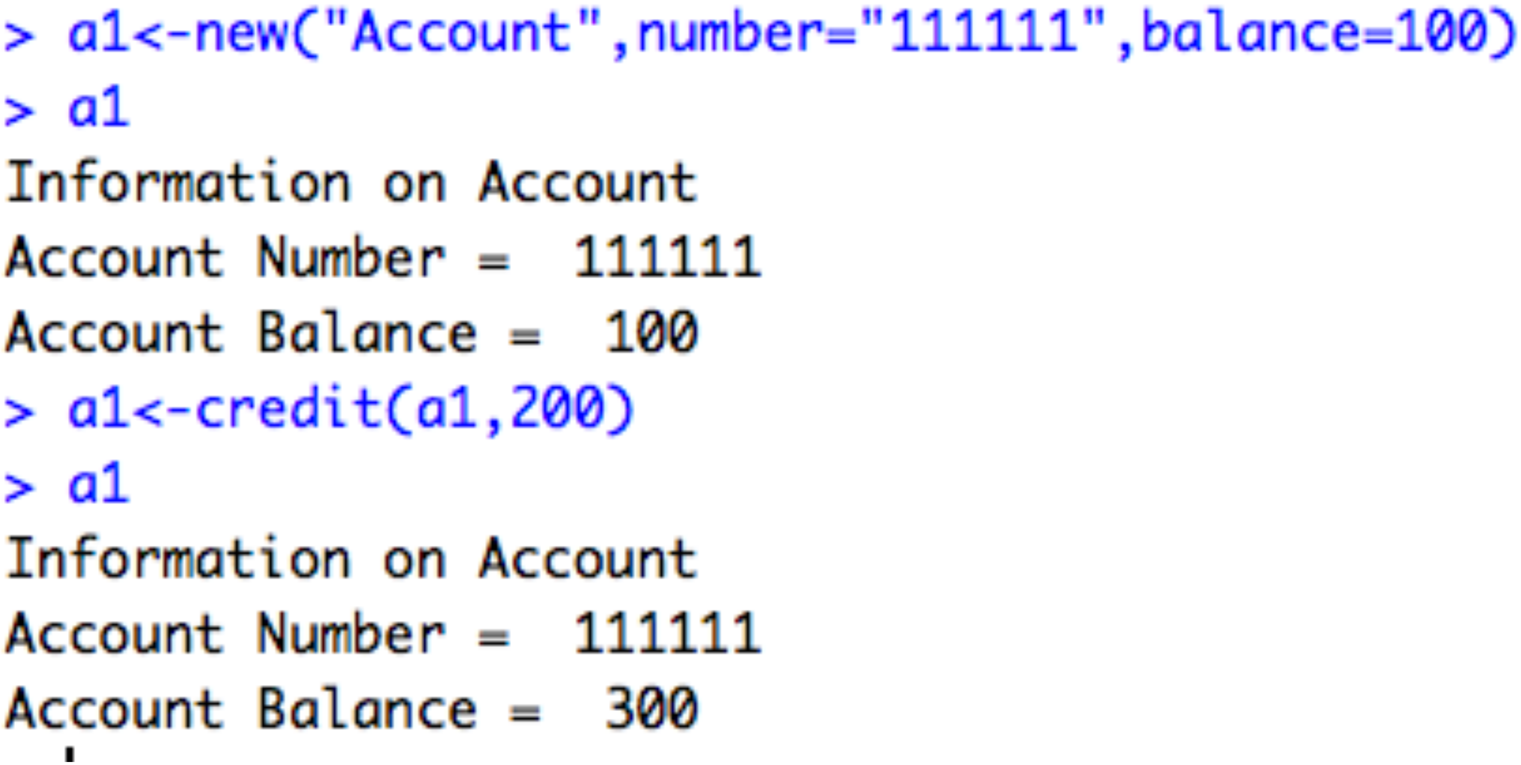
Description

Create and save a formal method for a given function and list of classes.

Usage

```
setMethod(f, signature=character(), definition,
          where = topenv(parent.frame()),
          valueClass = NULL, sealed = FALSE)
```

Invoking the new method



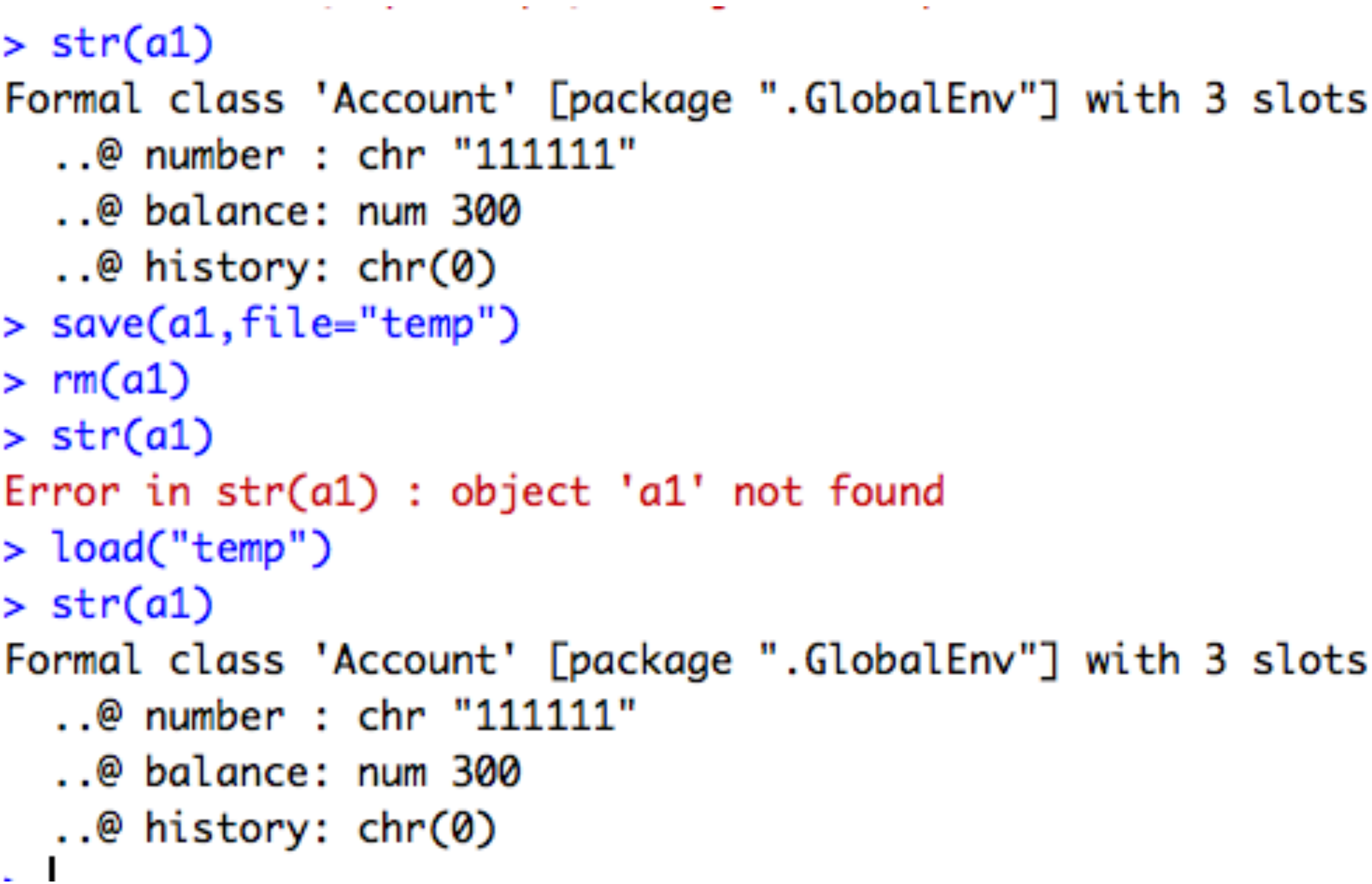
```
> a1<-new("Account",number="111111",balance=100)
> a1
Information on Account
Account Number =  111111
Account Balance =  100
> a1<-credit(a1,200)
> a1
Information on Account
Account Number =  111111
Account Balance =  300
```

S3 Versus S4 (Matloff 2010)

- Convenience of S3 vs Safety of S4
- John Chambers
  - S4 is needed in order to write “clear and reliable software”
- Google style guide:
  - “Use S3 objects and methods unless there is a strong reason to use S4 objects or methods. ”

http://google.github.io/styleguide/Rguide.xml

Saving objects to disk



```
> str(a1)
Formal class 'Account' [package ".GlobalEnv"] with 3 slots
  ..@ number : chr "111111"
  ..@ balance: num 300
  ..@ history: chr(0)
> save(a1,file="temp")
> rm(a1)
> str(a1)
Error in str(a1) : object 'a1' not found
> load("temp")
> str(a1)
Formal class 'Account' [package ".GlobalEnv"] with 3 slots
  ..@ number : chr "111111"
  ..@ balance: num 300
  ..@ history: chr(0)
```

Challenge 10.1

- Extend the example to include a transaction object for each account.
- This must have its own internal structure
- For every debit/credit, a transaction object must be created and stored.