Object Oriented Design(s) in R

Romain François

`romain@r-enthusiasts.com`

Chicago Local R User Group, Oct 27th, Chicago.

Outline

- Lexical Scoping
- S3 classes
- S4 classes
- Reference (R5) classes
- C++ classes
- Protocol Buffers

Fil rouge: bank account example

- ⋆ Data:
  - The balance
  - Authorized overdraft
- ⋆ Operations:
  - Open an account
  - Get the balance
  - Deposit
  - Withdraw


Lexical Scoping

```
> open.account <- function(total, overdraft = 0.0){
+     deposit <- function(amount) {
+         if( amount < 0 )
+             stop( "deposits must be positive" )
+         total <<- total + amount
+     }
+     withdraw <- function(amount) {
+         if( amount < 0 )
+             stop( "withdrawals must be positive" )
+         if( total - amount < overdraft )
+             stop( "you cannot withdraw that much" )
+         total <<- total - amount
+     }
+     balance <- function() {
+         total
+     }
+     list( deposit = deposit, withdraw = withdraw,
+         balance = balance )
+ }
> romain <- open.account(500)
> romain$balance()
[1] 500

> romain$deposit(100)
> romain$withdraw(200)
> romain$balance()
[1] 400
```


S3 classes

S3 classes

- Any R object with a **class** attribute
- Very easy
- Very dangerous
- Behaviour is added through S3 generic functions

```
> Account <- function( total, overdraft = 0.0 ){
+     out <- list( balance = total, overdraft = overdraft )
+     class( out ) <- "Account"
+     out
+ }
> balance <- function(x){
+     UseMethod( "balance" )
+ }
> balance.Account <- function(x) x$balance
```

S3 classes

```
> deposit <- function(x, amount){
+     UseMethod( "deposit" )
+ }
> deposit.Account <- function(x, amount) {
+     if( amount < 0 )
+         stop( "deposits must be positive" )
+     x$balance <- x$balance + amount
+     x
+ }
> withdraw <- function(x, amount){
+     UseMethod( "withdraw" )
+ }
> withdraw.Account <- function(x, amount) {
+     if( amount < 0 )
+         stop( "withdrawals must be positive" )
+     if( x$balance - amount < x$overdraft )
+         stop( "you cannot withdraw that much" )
+     x$balance <- x$balance - amount
+     x
+ }
```

S3 classes

Example use:

```
> romain <- Account( 500 )
> balance( romain )
[1] 500

> romain <- deposit( romain, 100 )
> romain <- withdraw( romain, 200 )
> balance( romain )
[1] 400
```

S4 classes

S4 classes

- Formal class definition
- Validity checking
- Formal generic functions and methods
- Very verbose, both in code and documentation

S4 classes

```
> setClass( "Account",
+     representation(
+         balance = "numeric",
+         overdraft = "numeric"
+     ),
+     prototype = prototype(
+         balance = 0.0,
+         overdraft = 0.0
+     ),
+     validity = function(object){
+         object@balance > object@overdraft
+     }
+ )
[1] "Account"

> setGeneric( "balance",
+     function(x) standardGeneric( "balance" )
+ )
[1] "balance"

> setMethod( "balance", "Account",
+     function(x) x@balance
+ )
[1] "balance"
```

S4 classes

```
> setGeneric( "deposit",
+     function(x, amount) standardGeneric( "deposit" )
+ )
[1] "deposit"

> setMethod( "deposit",
+     signature( x = "Account", amount = "numeric" ),
+     function(x, amount){
+         new( "Account" ,
+             balance = x@balance + amount,
+             overdraft = x@overdraft
+         )
+     }
+ )
[1] "deposit"
```

S4 classes

```
> romain <- new( "Account", balance = 500 )
> balance( romain )
[1] 500

> romain <- deposit( romain, 100 )
> romain <- withdraw( romain, 200 )
> balance( romain )
[1] 400
```

Reference (R5) classes

Reference (R5) classes

- Real S4 classes: formalism, dispatch, ...
- Passed by Reference
- Easy to use

```
> Account <- setRefClass( "Account_R5",
+     fields = list(
+         balance = "numeric",
+         overdraft = "numeric"
+     ),
+     methods = list(
+         withdraw = function( amount ){
+             if( amount < 0 )
+                 stop( "withdrawal must be positive" )
+             if( balance - amount < overdraft )
+                 stop( "overdrawn" )
+             balance <<- balance - amount
+         },
+         deposit = function(amount){
+             if( amount < 0 )
+                 stop( "deposits must be positive" )
+             balance <<- balance + amount
+         }
+     )
+ )
> x <- Account$new( balance = 10.0, overdraft = 0.0 )
> x$withdraw( 5 )
> x$deposit( 10 )
> x$balance
[1] 15
```

Real *pass by reference* :

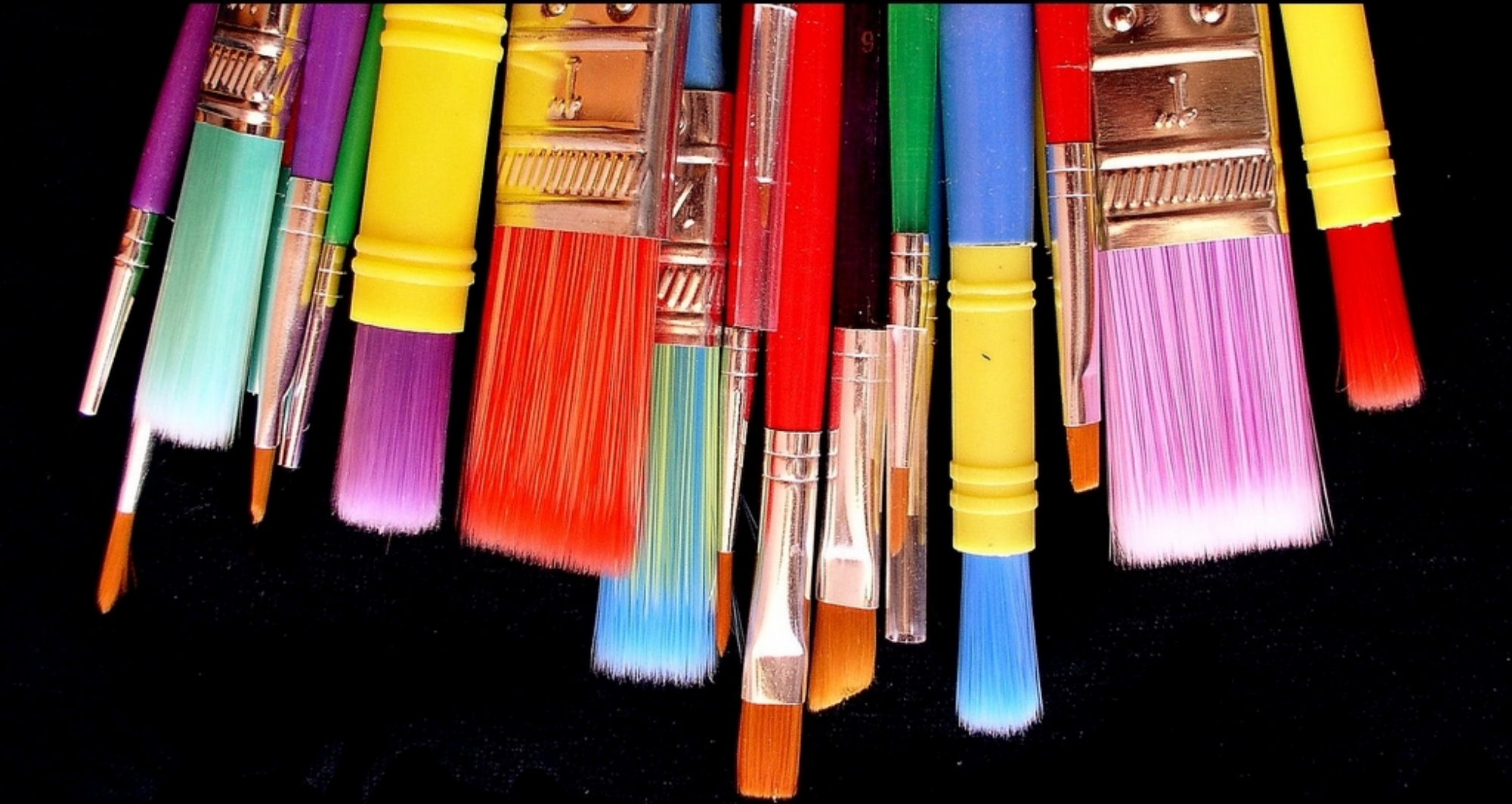
```
> borrow <- function( x, y, amount = 0.0 ){
+     x$withdraw( amount )
+     y$deposit( amount )
+     invisible(NULL)
+ }
> romain <- Account$new( balance = 5000, overdraft = 0.0 )
> dirk <- Account$new( balance = 3, overdraft = 0.0 )
> borrow( romain, dirk, 2000 )
> romain$balance
[1] 3000

> dirk$balance
[1] 2003
```

Adding a method dynamically to a class :

```
> Account$methods(
+     borrow = function(other, amount){
+         deposit( amount )
+         other$withdraw( amount )
+         invisible(NULL)
+     }
+ )
> romain <- Account$new( balance = 5000, overdraft = 0.0 )
> dirk <- Account$new( balance = 3, overdraft = 0.0 )
> dirk$borrow( romain, 2000 )
> romain$balance
[1] 3000

> dirk$balance
[1] 2003
```



C++ classes

C++ classes

```
class Account {
public:
    Account() :  balance(0.0), overdraft(0.0){}

    void withdraw( double amount ){
        if( balance - amount < overdraft )
            throw std::range_error( "no way") ;
        balance -= amount ;
    }

    void deposit( double amount ){
        balance += amount ;
    }

    double balance ;

private:
    double overdraft ;
} ;
```

C++ classes

Exposing to R through Rcpp modules:

```
RCPP_MODULE(yada){
    class_<Account>( "Account")

        // expose the field
        .field_readonly( "balance", &Account::balance )

        // expose the methods
        .method( "withdraw", &Account::withdraw )
        .method( "deposit", &Account::deposit ) ;

}
```

Use it in R:

```
> Account <- yada$Account
> romain <- Account$new()
> romain$deposit( 10 )
> romain$withdraw( 2 )
> romain$balance
[1] 8
```


Protocol Buffers

Protocol Buffers

Define the message type, in `Account.proto` :

```
package foo ;

message Account {
    required double balance = 1 ;
    required double overdraft = 2 ;
}
```

Load it into R with `RProtoBuf`:

```
> require( RProtoBuf )
> loadProtoFile( "Account.proto" )
```

Use it:

```
> romain <- new( foo.Account,
+     balance = 500, overdraft = 10 )
> romain$balance
```

Questions ?

Romain François
`http://romainfrancois.blog.free.fr`
`romain@r-enthusiasts.com`

Chicago Local R User Group, Oct 27th, Chicago.