Object-Oriented Programming in R

Christopher Eeles

2021-01-27

Section 1

What is Object-Oriented Programming?

Object-Oriented Programming

- Programming paradigm that organizes data and methods into classes
- One way to organize your code to make it easier to understand, reason about, and most importantly: reuse!

The Four Principles of OOP

- Encapsulation: objects manage their own state via methods
- Abstraction: hide complexity of implementation; only expose interfaces
- Inheritance: allows similar objects to share properties and methods
- Polymorphism: same method names, different method implementations

Why Do I Care?

- OOP can help group related data into a standard format
- OOP allows you to make valid (and guaranteed) assumptions about the structure of your data
- OOP allows you to build on the work of others by reusing or extending existing classes
- OOP allows you to write less code!
- OOP can abstract away input validation and reduce boiler plate error checking such as input validation
- Interfaces are awesome (more on this later)

Section 2

Generic Function OOP

What is Generic Function OOP?

- In generic function OOP, methods are first class objects
- They are NOT included in classes
- This allows modelling classes to be independent of modelling behaviour
  - What a thing is does not limit what it does
  - Much like in English, nouns are independent of verbs!

Generic Function OOP vs Message-Passing OOP

- The OOP system in R is very different from traditional message-passing OOP in languages like C++ or Java
- The methods available to a class are not in the definition of that class and are not set at compile time (indeed, in R there is no compile time)
- Instead, method dispatch is used to decide which implementation to call at runtime
- If there is no method implemented for that class; a method dispatch error is returned letting the user know

Benefits of Generic Function OOP

- In message-passing OOP, it is very easy to reuse nouns (classes)
- Consider a case where you like a class, but dislike the method implementation
- How is this resolved? Need to inherit from the class you like, define new methods using polymorphism; now you have a whole new class you have to maintain...
- A better solution: just define new verbs for the noun! This is possible in Generic Function OOP

Section 3

Bioconductor

Bioconductor

- Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data
  - This is where most of our R packages live! It is a great community with a ton of existing functionality to reuse
- The Bioconductor organization provides a rich set of S4 classes as core infrastructure in the Bioconductor development community!
  - If you weren't convinced on the value of S4 by any of my previous arguments, this one should be enough to change your mind

Bioconductor Packages

- BiocGenerics: implements and exports S4 generics for a wide variety of base R functions
  - If you are using the S4 system, you should check if a generic already exists here before defining your own
  - Establishes a standard set of verbs to operate on classes; equivalent operations of different classes should use the same verb
- S4Vectors: implementation of base R vectors in S4
  - Many abstract classes (interfaces)

Section 4

Using the S4 Object System

S4 Objects

- An S4 object is composed of a class name, one or more slots, and zero or more parent classes
- Slots can be accessed using the `@` operator; but this should only be used for development!
- S4 objects are just

Defining a Class

- Inheritance is accomplished using the `contains` parameter
- The new class contains all the slots from the old class
- New slots can be added using the `slots` argument

```
library(S4Vectors)
```

```
## Warning: package 'S4Vectors' was built under R version 4.0.

## Loading required package: stats4

## Loading required package: BiocGenerics

## Warning: package 'BiocGenerics' was built under R version 4

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
```

Defining a Constructor

- It is convention not to use the raw constructor object constructor, but instead defined a new method with better documentation of how the object gets built
- This isn't usually an S4 method; it is just a plain old R function

```
SumExpList <- function(...) {
    # make an object of the parent class
    simpleList <- SimpleList(...)

    # define the new requirements for your class
    mcols(simpleList)$class <-
        rep('SummarizedExperiment', length(as.list(...)))
    testSlot <- list('item'='This is a test slot!')

    # call your private constructor
    SElist <- .SumExpList(simpleList, testSlot=testSlot)
    return(SElist)
```

Class Validity

- This is the method that allows you to guarantee assumptions about your data!
- Implements a limited kind of static typing; since you know the type of some of your parameters, you don't need to check in your functions!

```
?setValidity
setValidity('SumExpList', function(object) {
    isSummarizedExperiment <-
        unlist(lapply(object@listData, is, class='SummarizedEx
    if (!all(isSummarizedExperiment)) {
        message('Items at indexes: ', paste0(which(!isSummariz
        ' are not SummarizedExperiments. All items in a SumExp
        'SummarizedExperiments!')
        return(FALSE)
    } else {
        return(TRUE)
    }
```

Class Validity

```
tryCatch({
    sumExpList <- SumExpList(list(item='Not a SummarizedExperi
}, error=function(e) message(e))
```

```
## Items at indexes: 1 are not SummarizedExperiments. All item
```

```
validSumExpList <- SumExpList(list(mySE=SummarizedExperiment()
```

Generic Functions

- Generic functions are template method definitions; they establish a verb that can be reused by others freely
- Generic functions are constrained only by the names of the arguments in the generic definition
  - Once you name an argument in a generic, it cannot be renamed
  - Unless you include the `...` parameter in the generic definition no new parameters can be specified in implementations (bad practice)

Defining a Generic

```
?setGeneric
# setGeneric('testSlot')
```

Method Dispatch

- Once a generic is defined, specific method implementations are defined based on the types of the arguments
- It is possible to define a different implementation based on the types of one or more parameters to the generic

Defining Methods

```
?setMethod
# setMethod('testSlot')
```

Section 5

References

OOP

- freeCodeComp: How to explain object-oriented programming concepts to a 6-year-old (https://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260/)
- educative: What is Object Oriented Programming? OOP Explained in Depth (https://www.educative.io/blog/object-oriented-programming)
- Wikipedia: Object-oriented programming (https://en.wikipedia.org/wiki/Object-oriented_programming)

S4

- Advanced R: The S4 object system (http://adv-r.had.co.nz/S4.html)
- Hansen, K.D. R - S4 Classes and Methods (https://kasperdanielhansen.github.io/genbioconductor/html/R_S4.html)
- Pages, H. (2016). A quick overview of the S4 class system (https://www.bioconductor.org/packages/release/bioc/vignettes/S4Vectors/inst/doc/S4QuickOverview.pdf)

Bioconductor

- BiocGenerics (https://bioconductor.org/packages/release/bioc/html/BiocGenerics.html)
- An Overview of the S4Vectors Package (https://bioconductor.org/packages/release/bioc/vignettes/S4Vectors/inst/doc/S4VectorsOverview.pdf)
-