Object-Oriented Programming in R

Kylie A. Bemis

Northeastern University
Khoury College of Computer Sciences

Northeastern University

Goals for this session

- What is OOP?
- S3
- S4

WHAT IS OOP?

"Everything that exists is an object"

- Every object in R has a ***class***
- An object's class describes its data type
- Some functions specialize for different classes

Specialized function behavior

```
plot(cars)
```

```
plot(cars$speed, cars$dist)
```

```
plot(dist ~ speed, data=cars)
```

The `plot()` function adapts to different types of arguments

What is OOP?

- **Object-oriented programming** (OOP) is a way of *organizing* data and code
- Built around the programming concepts of:
  - **Class** — A definition for a *type* of object
  - **Instance** — A *specific case* of that type of object
  - **Method** — Specialized *functions* for the object

Why use OOP?

- You've already been using OOP!
- Examples of **classes** in R/Bioconductor:
  - `data.frame`
  - `Matrix`
  - `SummarizedExperiment`
- Every object in R has a class

Features of OOP

- A *class* is a blueprint for organizing types of data
  - E.g., a `data.frame` stores tabular data in rows and columns
- An *instance* is a specific object of a class
  - E.g., `iris` and `mtcars` are instances of data.frame
- *Inheritance* allows subclasses to specialize superclasses
  - E.g., the tidyverse `tibble` inherits behavior from `data.frame`
- A *method* is a function specialized for a specific class
  - E.g., `plot()` is a *generic function* with methods for different classes

Generic functions

- R uses a functional-style of OOP
- Define a ***generic function*** to be specialized
  - E.g., `plot()` or `summary()`
- Methods belong to the generic
  - E.g., `plot.default()` or `summary.lm()`
- Call methods like ordinary functions

OOP in R

- S3 provides a simple system
  - No formal class definitions
  - Single dispatch (specialize on first argument only)
- S4 provides a more formal system
  - Formal class definitions and validity checking
  - Multiple dispatch (specialize on multiple arguments)

Using OOP in R

- Use S3 for simpler data structures
  - More common in base R and CRAN packages
- Use S4 for complex data structures
  - More common in Bioconductor packages

S3

S3 classes

- No formal class definition
- Create by setting `class` attribute
- Extends a base R data type
  - E.g., list, character, integer, double, etc.
- Methods named `generic.class()`

S3 example

```
Cat <- function(name) structure(list(name=name),
                                 class=c("Cat", "Animal"))
Dog <- function(name) structure(list(name=name),
                                 class=c("Dog", "Animal"))

speak.default <- function(object) print("*weird noises*")
speak.Animal <- function(object) print("*weird animal noises*")
speak.Cat <- function(object) print("Meow!")
speak.Dog <- function(object) print("Woof!")

mittens <- Cat("Mittens")
speak(mittens)
```

S4

S4 classes

- Formal class definition
- Create with `setClass()` function
- Define data *slots* of specific data types
  - Can define a validity method to check the object
- Methods created with `setMethod()`

S4 example

```
setClass("Animal4",
         contains = "VIRTUAL",
         slots = c(name = "character"),
         validity = function(object) {
           if ( length(object@name) != 1 )
             stop("slot 'name' must be length 1")
         })
setClass("Cat4", contains = "Animal4")
setClass("Dog4", contains = "Animal4")

Cat4 <- function(name) new("Cat4", name=name)
mittens4 <- Cat4("Mittens")

setMethod("speak", "Cat4", function(object) print("Meow!"))
speak(mittens4)
```

MS EXAMPLE

MSExample package

- Example package for working with raw mass spectra
- Implementation of basic S4 class for a mass spectrum
- Code available on Github:
  - https://github.com/kuwisdelu/MSExample
- Install via `remotes` package:
  - `remotes::install_github("kuwisdelu/MSExample")`

BREAK