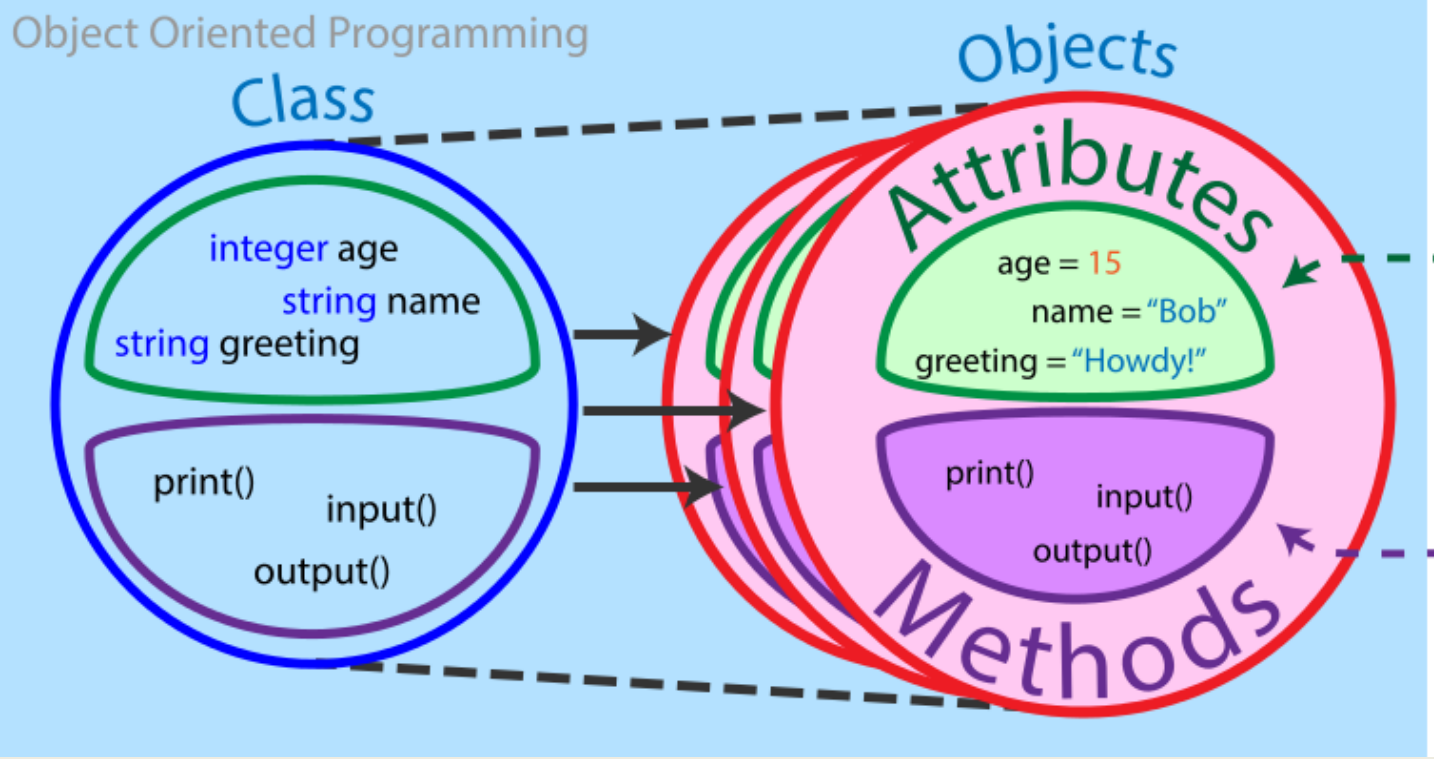
Reminder: proposals for final project

- Turn in Nov. 13 so I can give feedback before fall break (one paragraph or so)
- The project: make an R package with functions useful to your research
- Can work alone or in groups
- Plan for complexity of ~100 lines of code per person, plus documentation
- Optionally, you can let me know who will be responsible for what on group projects



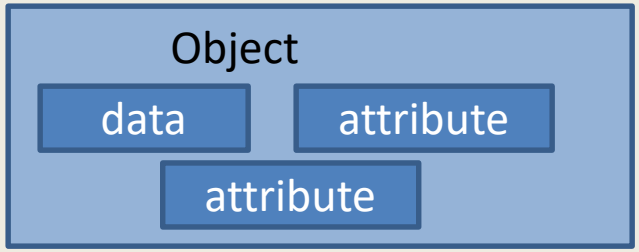
Basics of object-oriented programming (OOP)

Lecture 11, CPSC 499 Fall 2018

Image: Milesjpool, Wikimedia Commons

Adding attributes to an object in R

- We have seen these a bit already, for example `gregexpr` outputs vectors with a `match.length` attribute
- You can add or retrieve custom attributes with the `attr` function
- Allows you to store some metadata with your object
- Any type of object in R can become the attribute of another object

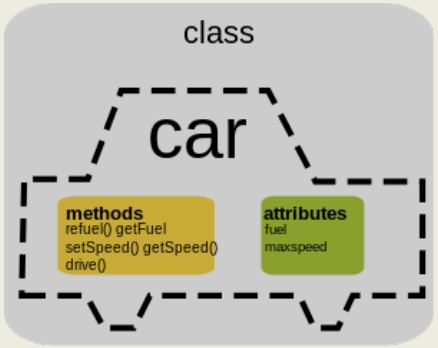


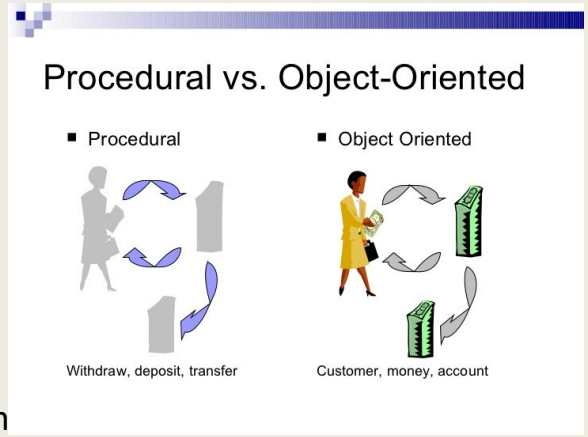
Maybe you want a more complex object…

- Several vectors, a matrix, etc. that are really all part of the same dataset
- You could make them all attributes of one object
- You might want a collection of functions for working with that particular type of dataset
- This is a great reason to use object-oriented programming

What is object-oriented programming?

- An object not only contains data/attributes, but also code (methods) that can be executed by the object
- The class of an object defines what sort of data it can contain and what methods it has available
- Imagine the object having an active rather than a passive role

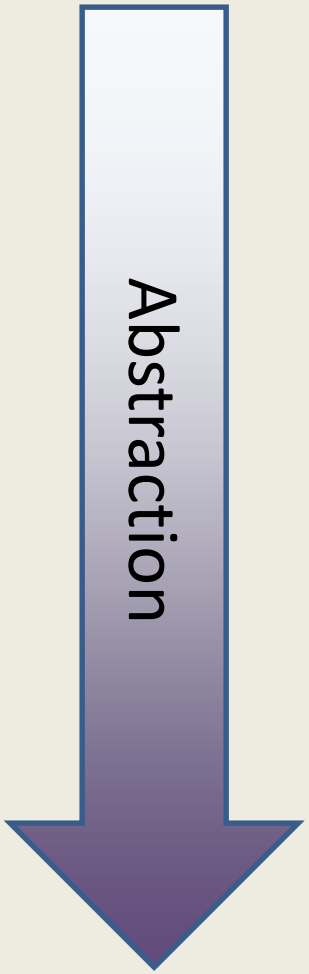




Images: litove.com and alphonsotech.com

Many languages allow a mix of programming styles

- **Procedural** programming: a series of steps that happen, all accessing the same global variables
- **Functional** programming: defining functions with their own environments/scopes
- **Object-oriented** programming: defining classes and methods



Abstraction in different programming styles

- Procedural
  - Variable name represents a value
  - Subroutine represents a series of steps
- Functional
  - Function represents a computation with a specific input and output, consistent behavior
- Object-oriented
  - Class represents a dataset of any complexity
  - Methods represent computations that can happen with that dataset

OOP terminology related to abstraction

- **Encapsulation**: Data and functions for manipulating it are bound together.
- **Data hiding**: Because the data are abstracted away, they can't be screwed up by another part of the program.
- **Data abstraction**: The user only sees the interface to the data. The implementation is hidden away.

Two systems in R for object-oriented programming

S3

- Informal
- You can assign any class name to any object
- Create methods just like creating functions, named `function.class`
- Better if you want a flexible class definition, able to change it as you go

S4

- Formal
- You have to create a class definition before you can make objects of that class
- More code involved in setting up methods
- More similar to object-oriented programming in other languages

Used heavily in Bioconductor

We'll focus on S3 since it is more practical for small projects

Constructor functions

- Function to create an object of a certain class
- The function has the same name as the class
- A constructor function
  - Puts the data into the object
  - Assigns the class name to the object
  - Adds any attributes
  - Checks to make sure data are formatted properly

Making new methods for existing generic functions

```
fn.cls <- function(object, ...){
  # normal code in here
}
```

Where `fn` is the name of the function and `cls` is the name of the class.

Common functions you may want methods for:

- print
- summary
- plot

Mini-exercise

- Make your own `summary` method for the "GPS" class

Defining new generic functions

```
fn <- function(object, ...){
  UseMethod("fn", object)
}
```

(Swap in the name of your function for “fn”.)

No additional code needed, just `UseMethod` to tell the function to look for a matching method.

Accessor functions

- Short functions that allow you to retrieve or assign data and attributes
- Add a level of abstraction between the user and the actual class structure
- Gives you flexibility to change the class structure without changing user interface

Making accessor functions

- Define as generic function and method
- Assignment functions check for errors, make sure the data conforms to what can go in the class

```
GetLoci <- function(object, ...){
  UseMethod("GetLoci", object)
}
GetLoci.RADdata <- function(object, ...){
  return(row.names(object$locTable))
}
```

I could change where locus names were stored in RADdata objects, then just change the GetLoci.RADdata method. All code that used GetLoci would still work.

```
GetContamRate <- function(object, ...){
  UseMethod("GetContamRate", object)
}
GetContamRate.RADdata <- function(object, ...){
  return(attr(object, "contamRate"))
}
SetContamRate <- function(object, value, ...){
  UseMethod("SetContamRate", object)
}
SetContamRate.RADdata <- function(object, value, ...){
  if(value < 0 || value > 1){
    stop("contamRate must range from zero to one.")
  }
  if(value > 0.01){
    warning("contamRate higher than expected.")
  }
  attr(object, "contamRate") <- value
  return(object)
}
```

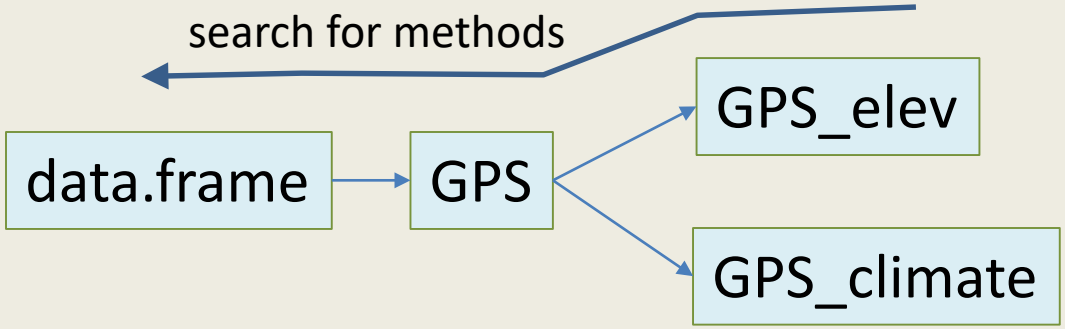
error checking

Mini-exercise

- Make a `GetDatum` accessor function to retrieve the datum attribute from a `GPS` object

Inheritance

- Why do we have a vector of two classes for `GPS`?
- By keeping `data.frame` as a superclass, we can still use `data.frame` methods for generic functions where we don't have `GPS` methods
- Search for methods starts at beginning of `class` vector and goes from there.



- data.frame is superclass of GPS
- GPS is subclass of data.frame
- GPS is superclass of GPS_elev
- etc.

Inheritance cont'd

- In addition to inheriting methods, subclasses generally inherit all the attributes (i.e. data slots) of their super classes
- S3 doesn't enforce this, but it is a good idea
- For example we could have a GPS_elev class that still had Lat and Long columns and a datum attribute, but also an Elevation column

Different structures for your class

- In the example just given we extended the data frame class.
- You can also make your own special classes of vectors or matrices
- **If you want your class to contain multiple objects, you can structure it as a list** (see http://www.cyclismo.org/tutorial/R/s3Classes.html for an example)

Putting S3 classes and methods into an R package

- Put your source code into the “R” directory like normal
- List any new generic functions within the `export` statement in NAMESPACE
- Add calls to S3method in NAMESPACE, e.g. `S3method(plot, GPS)`, for all new methods (both for existing generic functions and new generic functions)

Documenting S3 classes and methods

- To document the class, just document the constructor function like a normal function
- To document a method, format the alias and usage sections like so:

```
\name{AddGenotypeLikelihood}
\alias{AddGenotypeLikelihood}
\alias{AddGenotypeLikelihood.RADdata}
\title{
Estimate Genotype Likelihoods in a RADdata object
}
\description{
For each possible allele copy number across each possible ploidy in each taxon,
\code{AddGenotypeLikelihood} estimates the probability of observing the
distribution of read counts that are recorded for that taxon and locus.
}
\usage{
AddGenotypeLikelihood(object, ...)

\method{AddGenotypeLikelihood}{RADdata}(object, overdispersion = 9, \dots)
}
\arguments{
  \item{object}{
A \code{"\link{RADdata}"} object.
}
\item{overdispersion}{
An overdispersion parameter.  Higher values will cause the expected read depth
distribution to more resemble the binomial distribution.  Lower values indicate
more overdispersion, \emph{i.e.} sample-to-sample variance in the probability
of observing reads from a given allele.
}
  \item{\dots}{
Other arguments; none are currently used.
}
}
\details{
```

Alias for method

Usage for method

Arguments for generic function and method

Working with existing S4 classes

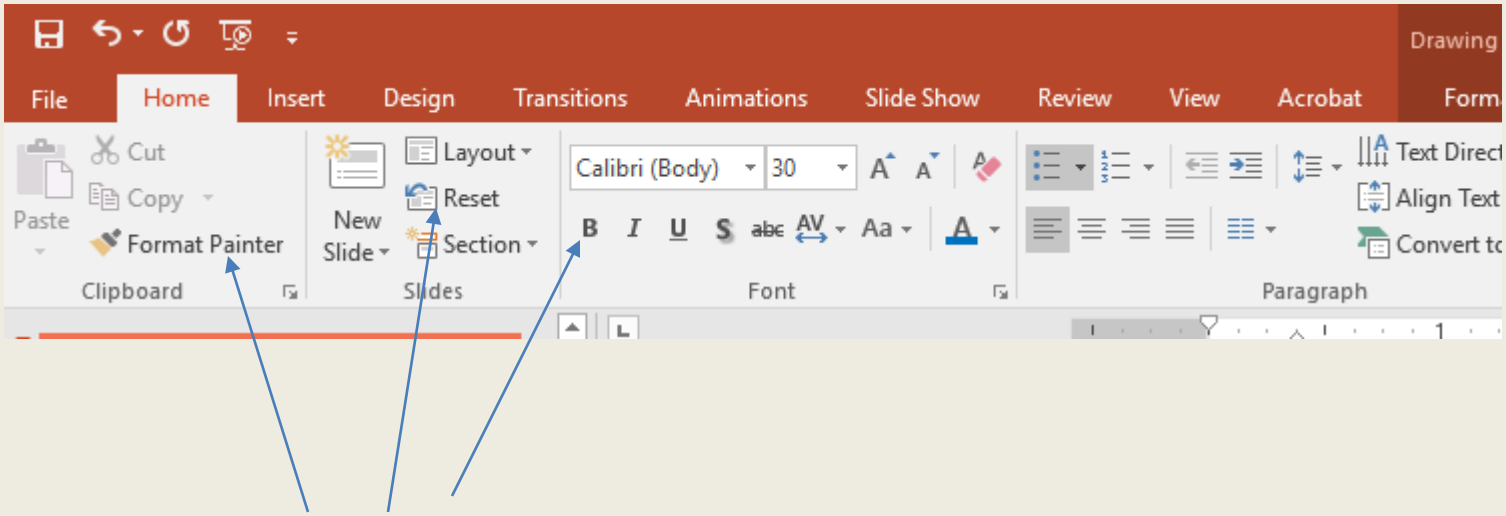
- If the class is designed well, there should be accessor functions for everything you need
- Additionally, slots can be accessed with the `@` operator

Reasons to learn/use S4 for making your own classes

- Lots of people are going to use your package
- You want to use inheritance in a more robust way
- You want to have replacement functions like
  - `Accessor(object) <- value`, not
  - `object <- SetValue(object, value)`
- You want very strict control over the type of data that can be contained in your class

Uses for OOP beyond R

- Used heavily by most languages for making graphical user interfaces (GUIs)



There is likely a “Button” class with many subclasses. Methods say what happen when the button is clicked. Every button is an object in the program.

Thursday's lab

- Set up an S3 class and some methods
- Incorporate into R package